CHAPTER 14

Text and Phonetic Analysis

Text-to-speech can be viewed as a speech coding system that yields an extremely high compression ratio coupled with a high degree of flexibility in choosing style, voice, rate, pitch range, and other playback effects. In this view of TTS, the text file that is input to a speech synthesizer is a form of coded speech. Thus, TTS subsumes coding technologies discussed in Chapter 7 with following goals:

- *Compression ratios superior to digitized wave files*—Compression yields benefits in many areas, including fast Internet transmission of spoken messages.
- *Flexibility in output characteristics*—Flexibility includes easy change of gender, average pitch, pitch range, etc., enabling application developers to give their systems' spoken output a unique individual personality. Flexibility also implies easy change of message content; it is generally easier to retype text than it is to record and deploy a digitized speech file.
- *Ability for perfect indexing between text and speech forms*—Preservation of the correspondence between textual representation and the speech wave form allows

synchronization with other media and output modes, such as word-by-word reverse video highlighting in a literacy tutor reading aloud.

- *Alternative access of text content*—TTS is the most effective alternative access of text for the blind, hands-free/eyes-free and displayless scenarios.

At first sight, the process of converting text into speech looks straightforward. However, when we analyze how complicated speakers read a text aloud, this simplistic view quickly falls apart. First, we need to convert words in written forms into speakable forms. This process is clearly nontrivial. Second, to sound natural, the system needs to convey the intonation of the sentences properly. This second process is clearly an extremely challenging one. One good analogy is to think how difficult to drop foreign accent when speaking a second language—a process still not quite understood by human beings.

The ultimate goal of simulating the speech of an understanding, effective human speaker from plain text is as distant today as the corresponding Holy Grail goals of the fields of speech recognition and machine translation. This is because such humanlike rendition depends on common-sense reasoning about the world and the text's relation to it, deep knowledge of the language itself in all its richness and variability, and even knowledge of the actual or expected audience—its goals, assumptions, presuppositions, and so on. In typical audio books or recordings for the visually challenged today, the human reader has enough familiarity with and understanding of the text to make appropriate choices for rendition of emotion, emphasis, and pacing, as well as handling both dialog and exposition. While computational power is steadily increasing, there remains a substantial knowledge gap that must be closed before fully human-sounding simulated voices and renditions can be created.

While no TTS system to date has approached optimal quality in the Turing test,[1] a large number of experimental and commercial systems have yielded fascinating insights. Even the relatively limited-quality TTS systems of today have found practical applications.

The basic TTS system architecture is illustrated in Chapter 1. In the present chapter we discuss text analysis and phonetic analysis whose objective is to convert words into speakable phonetic representation. The techniques discussed here are relevant to what we discussed for language modeling in Chapter 11 (like text normalization before computing *n*-gram) and for pronunciation modeling in Chapter 9. The next two modules—prosodic analysis and speech synthesis—are treated in the next two chapters.

14.1. Modules and Data Flow

The text analysis component, guided by presenter controls, is typically responsible for determining document structure, conversion of nonorthographic symbols, and parsing of language structure and meaning. The phonetic analysis component converts orthographic words to phones (unambiguous speech sound symbols). Some TTS systems assume dependency between text analysis, phonetic analysis, prosodic analysis, and speech synthesis, particu-

[1] [1] A test proposed by British mathematician Allan Turing of the ability of a computer to flawlessly imitate human performance on a given speech or language task [[29]].

larly systems based on very large databases containing long stretches of original, unmodified digitized speech with their original pitch contours. We discuss our high-level linguistic description of those modules, based on modularity, transparency, and reusability of components, although some aspects of text and phonetic analysis may be unnecessary for some particular systems.

We assume that the entire text (word, sentence, paragraph, document) to be spoken is contained in a single, wholly visible buffer. Some systems may be faced with special requirements for continuous flow-through or visibility of only small (word, phrase, sentence) *chunks* at a time, or extremely complex timing and synchronization requirements. The basic functional processes within the text and phonetic analysis are shown schematically in Figure 14.1.

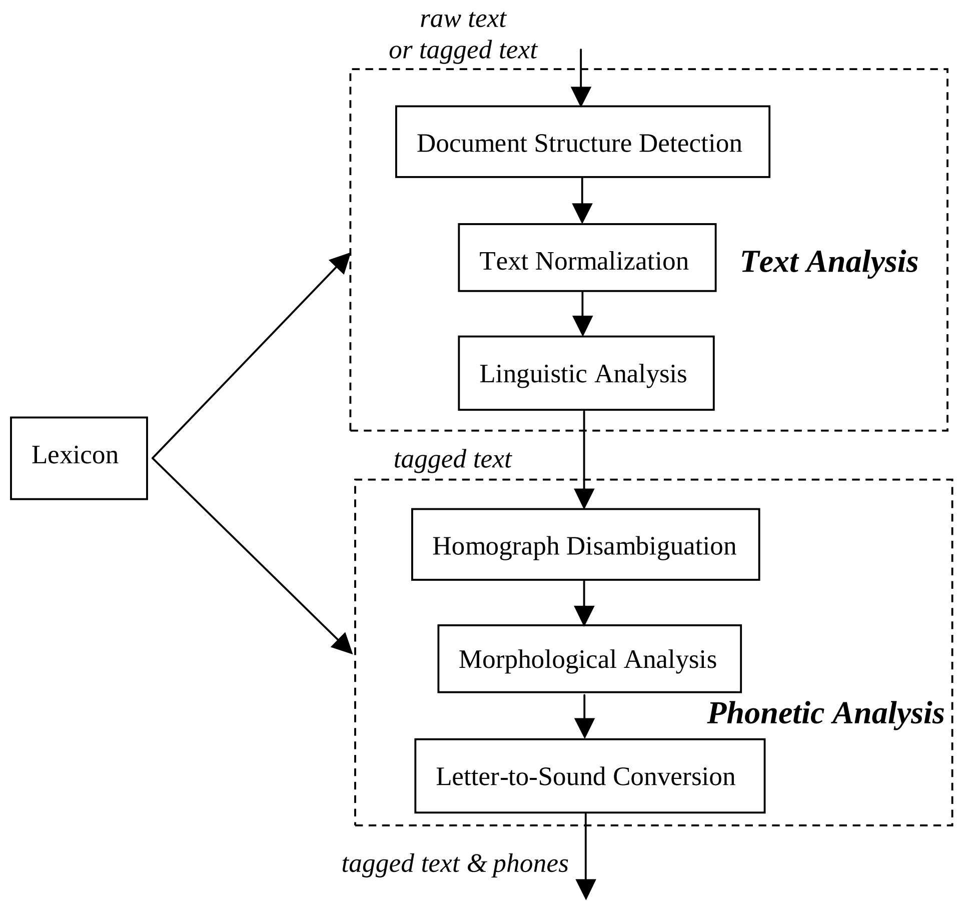

Figure 14.1 Modularized functional blocks for text and phonetic analysis components.

The architecture in Figure 14.1 brings the standard benefits of modularity and transparency. Modularity in this case means that the analysis at each level can be supplied by the most expert knowledge source, or a variety of different sources, as long as the markup conventions for expressing the analysis are uniform. Transparency means that the results of each stage could be reused by other processes for other purposes.

14.1.1. Modules

The *text analysis module* (TAM) is responsible for indicating all knowledge about the text or message that is not specifically phonetic or prosodic in nature. Very simple systems do little more than convert nonorthographic items, such as numbers, into words. More ambitious systems attempt to analyze whitespaces and punctuations to determine document structure, and perform sophisticated syntax and semantic analysis on sentences to determine attributes that help the phonetic analysis to generate correct phonetic representation and prosodic generation to construct superior pitch contours. As shown in Figure 14.1, text analysis for TTS involves three related processes:

- *Document structure detection*—Document structure is important to provide a context for all later processes. In addition, some elements of document structure, such as sentence breaking and paragraph segmentation, may have direct implications for prosody.
- *Text normalization*—Text normalization is the conversion from the variety symbols, numbers, and other nonorthographic entities of text into a common orthographic transcription suitable for subsequent phonetic conversion.
- *Linguistic analysis*—Linguistic analysis recovers the syntactic constituency and semantic features of words, phrases, clauses, and sentences, which is important for both pronunciation and prosodic choices in the successive processes.

The task of the phonetic analysis is to convert lexical orthographic symbols to phonemic representation along with possible diacritic information, such as stress placement. Phonetic analysis is thus often referred to grapheme-to-phoneme conversion. The purpose is obvious, since phonemes are the basic units of sound, as described in Chapter 2. Even though future TTS systems might be based on word sounding units with increasing storage technologies, homograph disambiguation and phonetic analysis for new words (either true new words being invented over time or morphologically transformed words) are still necessary for systems to correctly utter every word.

Grapheme-to-phoneme conversion is trivial for languages where there is a simple relationship between orthography and phonology. Such a simple relationship can be well captured by a handful of rules. Languages such as Spanish and Finnish belong to this category and are referred to as *phonetic languages*. English, on the other hand, is remote from phonetic language because English words often have many distinct origins. It is generally believed that the following three services are necessary to produce accurate pronunciations.

- *Homograph disambiguation*—It is important to disambiguate words with different senses to determine proper phonetic pronunciations, such as *object* (/*ah b jh eh k t*/) as a verb or as a noun (/*aa b jh eh k t*/).
- *Morphological analysis*—Analyzing the component morphemes provides important cues to attain the pronunciations for inflectional and derivational words.
- *Letter-to-sound conversion*—The last stage of the phonetic analysis generally includes general letter-to-sound rules (or modules) and a dictionary lookup to produce accurate pronunciations for any arbitrary word.

All the processes in text and phonetic analysis phases above need not to be deterministic, although most TTS systems today have deterministic processes. What we mean by *not deterministic* is that each of the above processes can generate multiple hypotheses with the hope that the later process can disambiguate those hypotheses by using more knowledge. For example, sometimes it might not be trivial to decide whether the punctuation "." is a sentence ending mark or abbreviation mark during document structure detection. The document structure detection process can pass both hypotheses to the later processes, and the decision can then be delayed until there is enough information to make an informed decision in later modules, such as the text normalization or linguistic analysis phases. When generating multiple hypotheses, the process can also assign probabilistic information if it comprehends the underlying probabilistic structure. This flexible pipeline architecture avoids the mistakes made by early processes based on insufficient knowledge.

Much of the work done by the text/phonetic analysis phase of a TTS system mirrors the processing attempted by *natural language process* (NLP) systems for other purposes, such as automatic proofreading, machine translation, database document indexing, and so on. Increasingly sophisticated NL analysis is needed to make certain TTS processing decisions in the examples illustrated in Table 14.1. Ultimately all decisions are context driven and probabilistic in nature, since, for example, dogs might be cooked and eaten in some cultures.

Table 14.1 Examples of several ambiguous text normalization cases.

Examples	Alternatives	Techniques
Dr. Smith	*doctor* or *drive*?	abbreviation analysis, case analysis
Will you go?	yes-no or wh-question?	syntactic analysis
I ate a hot dog.	accent on *dog*?	semantic, verb/direct object likelihood
I saw a hot dog.	accent on *dog*?	discourse, pragmatic analysis

Most TTS systems today employ specialized natural language processing modules for front-end analysis. In the future, it is likely that less emphasis will be placed on construction of TTS-specific text/phonetic analysis components such as those described in [27], while more resources will likely go to general-purpose NLP systems with cross-functional potential [23]. In other words, all the modules above only perform simple processing and pass all possible hypotheses to the later modules. At the end of the text/phonetic phase, a unified NLP module then performs extensive syntactic/semantic analysis for the best decisions. The necessity for such an architectural approach is already visible in markets where language issues have forced early attention to common lexical and tokenization resources, such as Japan. Japanese system services and applications can usually expect to rely on common cross-functional linguistic resources, and many benefits are reaped, including elimination of bulk, reduction of redundancy and development time, and enforcement of systemwide consistent behavior. For example, under Japanese architectures, TTS, recognition, sorting, word processing, database, and other systems are expected to share a common language and dictionary service.

14.1.2. Data Flows

It is arguable that text input alone does not give system enough information to express and render the intention of the text producer. Thus, more and more TTS systems focus on providing an infrastructure of standard set of markups (tags), so that the text producer can better express their semantic intention with these markups in addition to plain text. These kinds of markups have different levels of granularity, ranging from simple speed settings specified in *words per minute* up to elaborate schemes for semantic representation of concepts that may bypass the ordinary text analysis module altogether.[2] The markup can be done by internal proprietary conventions or by some standard markup, such as XML (Extensible Markup Language [35]). Some of these markup capabilities will be discussed in Sections 14.3 and 14.4.

For example, an application may know a lot about the structure and content of the text to be spoken, and it can apply this knowledge to the text, using common markup conventions, to greatly improve spoken output quality. On the other hand, some applications may have certain broad requirements such as rate, pitch, callback types, etc. For engines providing such supports, the text and/or phonetic analysis phase can be skipped, in whole or in part. Whether the application or the system has provided the text analysis markup, the structural conventions should be identical and must be sufficient to guide the phonetic analysis. The phonetic analysis module should be presented only with markup tags indicating structure or functions of textual chunks, and words in standard orthography. The similar phonetic markups could also be presented to the phonetic analysis module, the module could be skipped.

Internal architectures, data structures, and interfaces may vary widely from system to system. However, most modern TTS systems initially construct a simple description of an utterance or paragraph based on observable attributes, typically text words and punctuation, perhaps augmented by control annotations. This minimal initial *skeleton* is then augmented with many layers of structure hypothesized by the TTS system's internal analysis modules. Beginning with a surface stream of words, punctuation, and other symbols, typical layers of detected structure that may be added include:

- Phonemes

[2] This latter type of system is sometimes called *concept-to-speech* or *message-to-speech*, which is described in Chapter 17. It generally generates better speech rendering when domain-specific knowledge is provided to the system.

- Syllables
- Morphemes
- Words derived from nonwords (such as dates like “9/10/99”)
- Syntactic constituents
- Relative importance of words and phrases
- Prosodic phrasing
- Accentuation
- Duration controls
- Pitch controls

We can now consider how the information needed to support synthesis of a sentence is developed in processing an example sentence such as: “A skilled electrician reported.”

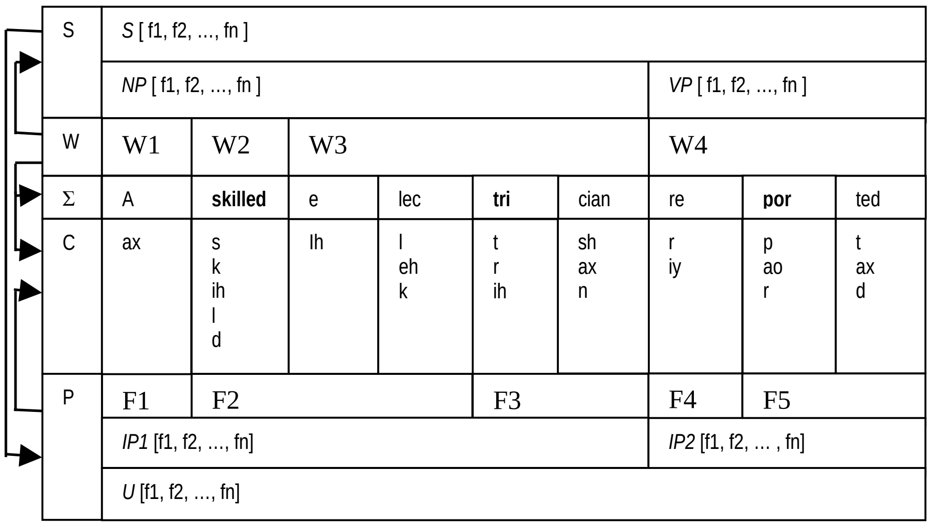

Figure 14.2 Annotation tiers indicating incremental analysis based on an input (text) sentence “A skilled electrician reported.” Flow of incremental annotation is indicated by arrows on the left side.

In Figure 14.2, the information that must be inferred from text is diagrammed. The flow proceeds as follows:

- **W(ords) ➔ Σ, C(ontrols)**: the syllabic structure (Σ) and the basic phonemic form of a word are derived from lexical lookup and/or the application of rules. The Σ tier shows the syllable divisions (written in text form for convenience). The C tier, at this stage, shows the basic phonemic symbols for each word’s syllables.

- **W(ords) ➔ S(yntax/semantics)**: The word stream from text is used to infer a syntactic and possibly semantic structure (S tier) for an input sentence. Syntactic and semantic structure above the word would include syntactic constituents such as Noun Phrase (NP), Verb Phrase (VP), etc. and any semantic features that can be recovered from the current sentence or analysis of other contexts that may be available (such as an entire paragraph or document). The lower-level phrases such as NP and VP may be grouped into broader constituents such as Sentence (S), depending on the parsing architecture.
- **S(yntax/semantics) ➔ P(rosody)**: The P(rosodic) tier is also called the symbolic prosodic module. If a word is semantically *important* in a sentence, that importance can be reflected in speech with a little extra phonetic prominence, called an accent. Some synthesizers begin building a prosodic structure by placing metrical foot boundaries to the left of every accented syllable. The resulting metrical foot structure is shown as F1, F2, etc. in Figure 14.2 (some *feet* lack an accented head and are 'degenerate'). Over the metrical foot structure, higher-order prosodic constituents, with their own characteristic relative pitch ranges, boundary pitch movements, etc. can be constructed, shown in the figure as intonational phrases IP1, IP2. The details of prosodic analysis, including the meaning of those symbols, are described in Chapter 15.

The final phonetic form of the words to be spoken will reflect not only the original phonetics, but decisions made in the S and P tiers as well. For example, the P(rosody) tier adds detailed pitch and duration controls to the C(ontrol) specification that is passed to the voice synthesis component. Obviously, there can be a huge variety of particular architectures and components involved in the conversion process. Most systems, however, have some analog to each of the components presented above.

14.1.3. Localization Issues

A major issue in the text and phonetic analysis components of a TTS system is internationalization and localization. While most of the language processing technologies in this book are exemplified by English case studies, an internationalized TTS architecture enabling minimal expense in localization is highly desirable. From a technological point of view, the text conventions and writing systems of language communities may differ substantially in arbitrary ways, necessitating serious effort in both specifying an internationalized architecture for text and phonetic analysis, and localizing that architecture for any particular language.

For example, in Japanese and Chinese, the unit of *word* is not clearly identified by spaces in text. In French, interword dependencies in pronunciation realization exist (liaison). Conventions for writing numerical forms of dates, times, money, etc. may differ across languages. In French, number groups separated by spaces may need to be integrated as single amounts, which rarely occurs in English. Some of these issues may be more serious for certain types of TTS architectures than others. In general, it is best to specify a rule architecture for text processing and phonetic analysis based on some fundamental formalism that allows for language-particular data tables, and which is powerful enough to handle a wide range of relations and alternatives.

14.2. LEXICON

The most important resource for text and phonetic analysis is the TTS system lexicon (also referred to as a dictionary). As illustrated in Figure 14.1, the TTS system lexicon is shared with almost all components. The lexical service should provide the following kinds of content in order to support a TTS system:

- Inflected forms of lexicon entries
- Phonetic pronunciations (support multiple pronunciations), stress and syllabic structure features for each lexicon entry
- Morphological analysis capability
- Abbreviation and acronym expansion and pronunciation
- Attributes indicating word status, including proper-name tagging, and other special properties
- List of speakable names of all common single characters. Under modern operating systems, the characters should include all Unicode characters.
- Word part-of-speech (POS) and other syntactic/semantic attributes
- Other special features, e.g., how likely a word is to be accented, etc.

It should be clear that the requirements for a TTS system lexical service overlap heavily with those for more general-purpose NLP.

Traditionally, TTS systems have been rule oriented, in particular for grapheme-to-phoneme conversion. Often, tens of so called *letter-to-sound* (LTS) rules (described in detail in Section 14.8) are used first for grapheme-to-phoneme conversion, and the role of the lexicon has been minimized as an *exception list*, whose pronunciations cannot be predicted on the basis of such LTS rules. However, this view of the lexicon's role has increasingly been adjusted as the requirement of a sophisticated NLP analysis for high-quality TTS systems has become apparent. There are a number of ways to optimize a dictionary system. For a good overview of lexical organization issues, please see [4].

To expose different contents about a lexicon entry listed above for different TTS module, it calls for a consistent mechanism. It can be done either through a database query or a function call in which the caller sends a key (usually the orthographic representation of a word) and the desired attribute. For example, a TTS module can use the following function call to look up a particular attribute (like phonetic pronunciations or POS) by passing the attribute *att* and the result will be stored in the pointer *val* upon successful lookup. Moreover, when the lookup is successful (the word is found in the dictionary) the function returns true, otherwise it will return false instead.

```
BOOLEAN    DictLookup (string word, ATTTYPE att, (VOID *) val)
```

We should also point out that this functional view of dictionary could further expand the physical dictionary as a service. The morphological analysis and letter-to-sound modules (described in Sections 14.7 and 14.8) can all be incorporated into the same lexical service. That is, underneath dictionary lookup, operation and analysis is encapsulated from users to form a uniform service.

Another consideration in the system's runtime dictionary is compression. While many standard compression algorithms exist, and should be judiciously applied, the organization and extent of the vocabulary itself can also be optimized for small space and quick search. The kinds of American English vocabulary relevant to a TTS system include:

- Grammatical function words (closed class)—about several hundred)
- Very common vocabulary—about 5,000 or more
- College-level core vocabulary base forms—about 60,000 or more
- College-level core vocabulary inflected form—about 120,000 or more
- Scientific and technical vocabulary, by field—e.g., legal, medical, engineering, etc.
- Personal names—e.g., family, given, male, female, national origin, etc.
- Place names—e.g., countries, cities, rivers, mountains, planets, stars, etc.
- Slang
- Archaisms

The typical sizes of reasonably complete lists of the above types of vocabulary run from a few hundred function or closed-class words (such as prepositions and pronouns) to 120,000 or so inflected forms of college-level vocabulary items, up to several million surnames and place names. Careful analysis of the likely needs of typical target applications can potentially reduce the size of the runtime dictionary. In general, most TTS systems maintain a system dictionary with a size between 5000 and 200,000 entries. With advanced technologies in database and hashing, search is typically a nonissue for dictionary lookup. In addition, since new forms are constantly produced by various creative processes, such as acronyms, borrowing, slang acceptance, compounding, and morphological manipulation, some means of analyzing words that have not been stored must be provided. This is the topic of Sections 14.7 and 14.8.

14.3. DOCUMENT STRUCTURE DETECTION

For the purpose of discussion, we assume that all input to the TAM is an XML document, though perhaps largely unmarked, and the output is also a (more extensively marked) XML document. That is to say, all the knowledge recovered during the TAM phase is to be expressed as XML markup. This confirms the independence of the TAM from phonetic and prosodic considerations, allowing a variety of resources, some perhaps not crafted with TTS in mind, to be brought to bear by the TAM on the text. It also implies that that output of the TAM is potentially usable by other, non-TTS processes, such as normalization of language-model training data for building statistical language models (see Chapter 11). This fully modular and transparent view of TTS allows the greatest flexibility in document analysis, provides for direct *authoring* of structure and other customization, while allowing a split between expensive, multipurpose natural language analysis and the core TTS functionality. Although other text format or markup language, such as Adobe Acrobat or Microsoft Word, can be used for the same purpose, the choice of XML is obvious because it is the widely open standard, particularly for the Internet.

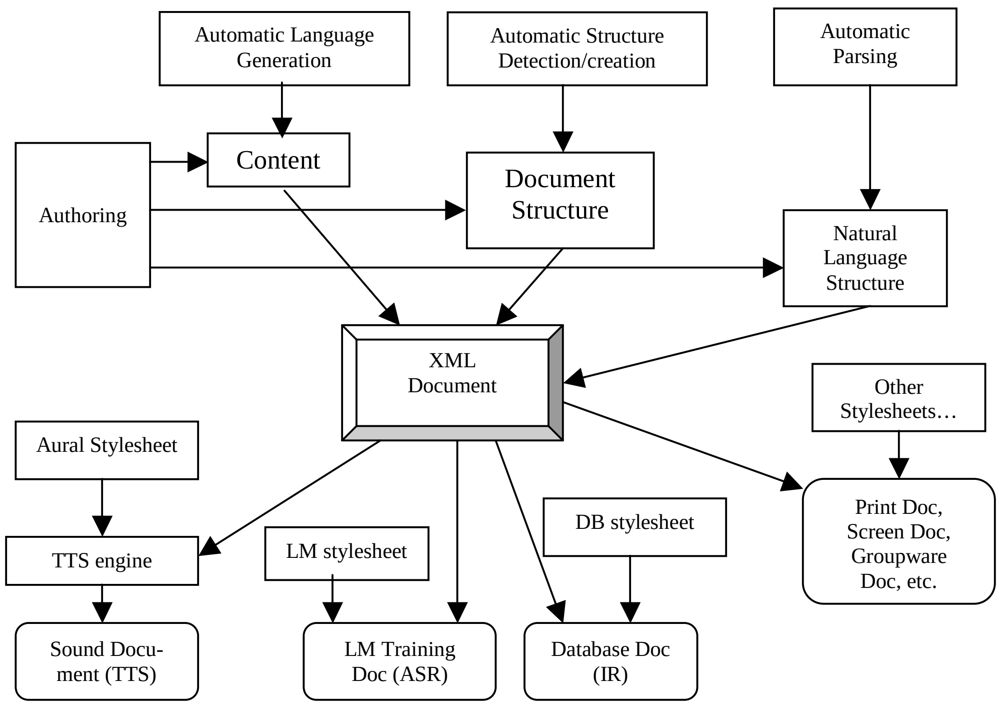

Figure 14.3 A documentcentric view of TTS.

XML is a set of conventions for indicating the semantics and scope of various entities that combine to constitute a document. It is conceptually somewhat similar to Hypertext Markup Language (HTML), which is the exchange code for the World Wide Web. In these markup systems, properties are identified by tags with explicit scope, such as "`<b>make this phrase bold</b>`" to indicate a heavy, dark print display. XML in particular attempts to enforce a principled separation between document structure and content, on one hand, and the detailed formatting or presentation requirements of various *uses* of documents, on the other. Since we cannot provide a tutorial on XML here, we freely introduce example tags that indicate document and linguistic structure. The interpretations of these are intuitive to most readers, though, of course, the analytic knowledge underlying decisions to insert tags may be very sophisticated. It will be some time before commercial TTS engines come to a common understanding on the wide variety of text attributes that should be marked, and accept a common set of conventions. Nevertheless, it is reasonable to adopt the idea that TAM should be independent and reusable, thus allowing XML documents (which are expected to proliferate) to function for speech just as for other modalities, as indicated schematically in Figure 14.3.

TTS is regarded in Figure 14.3as a factored process, with the text analysis perhaps carried out by human editors or by natural language analysis systems. The role of the TTS engine per se may eventually be reduced to the interpretation of structural tags and provision of phonetic information. While commercial engines of the present day are not structured with these assumptions in mind, modularity and transparency are likely to become increasingly important. The increasing acceptance of the basic ideas underlying an XML documentcentric approach to text and phonetic analysis for TTS can be seen in the recent proliferation of XML-like speech markup proposals [24, 33]. While not presenting any of these in detail, in the discussion below we adopt informal conventions that reflect and extend their basic assumptions. The structural markup exploited by the TTS systems of the future may be imposed by XML authoring systems at document creation time, or may be inserted by independent analytical procedures. In any case the distinction between purely automatic structure creation/detection and human annotation and authoring will increasingly blur—just as in natural language translation and information retrieval domains, the distinction between machine-produced results and human-produced results has begun to blur.

14.3.1. Chapter and Section Headers

Section headers are a standard convention in XML document markup, and TTS systems can use the structural indications to control prosody and to regulate prosodic style, just as a professional reader might treat chapter headings differently. Increasingly, a document created on computer or intended for any kind of electronic circulation incorporates structural markup, and the TTS and audio human-computer-interface systems of the future learn to exploit this (in longer documents, the document structure markup assists in audio navigation, speedup, and skipping). For example, the XML annotation of a book at a high level might follow conventions as shown in Figure 14.4. Viewing a document in this way might lead a TTS system to insert pauses and emphasis correctly, in accordance with the structure marked. Furthermore, an audio interface system would work jointly with a TTS system to allow easy navigation and orientation within such a structure. If future documents are marked up in this fashion, the concept of audio books, for example, would change to rely less on unstructured prerecorded speech and more on smart, XML-aware, high-quality audio navigation and TTS systems, with the output customization flexibility they provide.

For documents without explicit markup information for section and chapter headers, it is in general a nontrivial task to detect them automatically. Therefore, most TTS systems today do not make such an attempt.

```
<Book>
        <Title>The Pity of War</Title>
        <Subtitle>Explaining World War I</Subtitle>
        <Author>Niall Ferguson</Author>
        <TableOfContents>…</TableOfContents>
        <Introduction>
                <Para>…</Para>
                …
        </Introduction>
        <Chapter>
        <ChapterTitle>The Myths of Militarism</ChapterTitle>
                <Section>
                        <SectionTitle>Prophets</SectionTitle>
                        <Para> … </Para>
                        …
                </Section>
        </Chapter>
        …
</Book>
```

Figure 14.4 An example of the XML annotation of a book.

14.3.2. Lists

Lists or bulleted items may be rendered with distinct intonational contours to indicate aurally their special status. This kind of structure might be indicated in XML as shown in Figure 14.5. Again, TTS engine designers need to get used to the idea of accepting such markup for interpretation, or incorporating technologies that can detect and insert such markup as needed by the downstream phonetic processing modules. Similar to chapter and section headers, most TTS systems today do not make an attempt to detect list structures automatically.

```
<UL>
<LI>compression</LI>
<LI>flexibility</LI>
<LI>text-waveform correspondence</LI>
</UL>
<Caption>The advantages of TTS</Caption>
```

Figure 14.5 An example of a list marked by XML.

14.3.3. Paragraphs

The paragraph has been shown to have direct and distinctive implications for pitch assignment in TTS [26]. The pitch range of good readers or speakers in the first few clauses at the start of a new paragraph is typically substantially higher than that for mid-paragraph sentences, and it narrows further in the final few clauses, before resetting for the next paragraph. Thus, to mimic a high-quality reading style in future TTS systems, the paragraph structure has to be detected from XML tagging or inferred from inspection of raw formatting. Obviously, relying on independently motivated XML tagging is, as always, the superior option, especially since this is a very common structural annotation in XML documents.

In contrast to other document structure information, paragraphs are probably among the easiest to detect automatically. The character <CR> (carriage return) or <NL> (new line) is usually a reliable clue for paragraphs.

14.3.4. Sentences

While sentence breaks are not normally indicated in XML markup today, there is no reason to exclude them, and knowledge of the sentence unit can be crucial for high-quality TTS. In fact, some XML-like conventions for text markup of documents to be rendered by synthesizers (e.g., SABLE) provide for a DIV (division) tag that could take paragraph, sentence, clause, etc. as attribute [24]. If we define sentence broadly as a primal linguistic unit that makes up paragraphs, attributes could be added to a `Sent` tag to express whatever linguistic knowledge exists about the type of the sentence as a whole:

<Sent type="yes-no question">
Is life so dear, or peace so sweet, as to be purchased at the price of chains and slavery?
</Sent>

Again, as emphasized throughout this section, such annotation could be either applied during creation of the XML documents (of the future) or inserted by independent processes. Such structure-detection processes may be motivated by a variety of needs and may exist outside the TTS system per se.

If no independent markup of sentence structure is available from an external, independently motivated document analysis or natural language system, a TTS system typically relies on simple internal heuristics to guess at sentence divisions. In e-mail and other relatively informal written communications, sentence boundaries may be very hard to detect. In contrast to English, sentence breaking could be trivial for some other written languages. In Chinese, there is a designated symbol (a small circle □) for marking the end of a sentence, so the sentence breaking could be done in a totally straightforward way. However, for most Asian languages, such as Chinese, Japanese, and Thai, there is in general no space within a sentence. Thus, tokenization is an important issue for Asian languages.

In more formal English writing, sentence boundaries are often signaled by terminal punctuation from the set: `{.!?}` followed by whitespaces and an upper-case initial word. Sometimes additional punctuation may trail the '?' and '!' characters, such as close quotation marks and/or close parenthesis. The character '.' is particularly troubling, because it is, in programming terms, heavily *overloaded*. Apart from its uses in numerical expressions and internet addresses, its other main use is as a marker of abbreviation, itself a difficult problem for text normalization (see Section 14.4). Consider this pathological jumble of potentially ambiguous cases:

> Mr. Smith came by. He knows that it costs $1.99, but I don't know when he'll be back (he didn't ask, "when should I return?")… His website is www.mrsmithhhhhh.com. The car is 72.5 in. long (we don't know which parking space he'll put his car in.) but he said "…and the truth shall set you free," an interesting quote.

Some of these can be resolved in the linguistic analysis module. However for some cases, only probabilistic guesses can be made, and even a human reader may have difficulty. The ambiguous sentence breaking can also be resolved in an abbreviation-processing module (described in Section 14.4.1). Any period punctuation that is not taken to signal an abbreviation and is not part of a number can be taken as end-of-sentence. Of course, as we have seen above, abbreviations are also confusable with words that can naturally end sentences, e.g., "*in*." For the measure abbreviations, an examination of the left context (checking for numeric) may be sufficient. In any case, the complexity of sentence breaking illustrates the value of passing multiple hypotheses and letting later, more knowledgeable modules (such as an abbreviation or linguistic analysis module) make decisions. Algorithm 14.1 shows a simple sentence-breaking algorithm that should be able to handle most cases.

ALGORITHM 14.1 A SIMPLE SENTENCE-BREAKING ALGORITHM

```
1. If found punctuation ./!/? advance one character and goto 2.
   else advance one character and goto 1.
2. If not found whitespace advance one character and goto 1.
3. If the character is period (.) goto 4.
   else goto 5.
4. Perform abbreviation analysis.
     If not an abbreviation goto 5.
     else advance one character and goto 1.
5. Declare a sentence boundary and sentence type ./!/?
   Advance one character and goto 1.
```

For advanced sentence breakers, a weighted combination of the following kinds of considerations may be used in constructing algorithms for determining sentence boundaries (ordered from easiest/most common to most sophisticated):

- Abbreviation processing—Abbreviation processing is one of the most important tasks in text normalization and will be described in detail in Section 14.4.
- Rules or CART built (Chapter 4) upon features based on: document structure, whitespace, case conventions, etc.

- Statistical frequencies on sentence-initial word likelihood
- Statistical frequencies of typical lengths of sentences for various genres
- Streaming syntactic/semantic (linguistic) analysis—Syntactic/semantic analysis is also essential for providing critical information for phonetic and prosodic analysis. Linguistic analysis will be described in Section 14.5.

As you can see, a deliberate sentence breaking requires a fair amount of linguistic processing, like abbreviation processing and syntactic/semantic analysis. Since this type of analysis is typically included in the later modules (text normalization or linguistic analysis), it might be a sensible decision to delay the decision for sentence breaking until later modules, either text normalization or linguistic analysis. In effect, this arrangement can be treated as the document structure module passing along multiple hypotheses of sentence boundaries, and it allows later modules with deeper linguistic knowledge (text normalization or linguistic analysis) to make more intelligent decisions.

Finally, if a long buffer of unpunctuated words is presented, TTS systems may impose arbitrary limits on the length of a sentence for later processing. For example, the writings of the French author Marcel Proust contain some sentences that are several hundred words long (average sentence length for ordinary prose is about 15 to 25 words).

```
<message>
    <header>
            <date>11 June 1998</date>
            <from>Leslie</from>
            <to>Jo</to>
            <subject>Surf’s Up!</subject>
    </header>
    <body> … </body>
    <sig>Freedom’s just another word for nothing left to lose</sig>
</message>
```

Figure 14.6 An example of e-mail marked by XML.

14.3.5. E-mail

TTS could be ideal for reading e-mail over the phone or in an eyes-busy situation such as when driving a motor vehicle. Here again we can speculate that XML-tagged e-mail structure, minimally something like the example in Figure 14.6, will be essential for high-quality prosody, and for controlling the audio interface, allowing skips and speedups of areas the user has defined as less critical, and allowing the system to announce the function of each block. For example, the `sig` (signature) portion of e-mail certainly has a different semantic function than the main message text and should be clearly identified as such, or skipped, at the listener’s discretion. Modern e-mail systems are providing increasingly sophisticated support for structure annotation such as that exemplified in Figure 14.6. Obviously, the e-mail document structure can be detected only with appropriate tags (like XML). It is very difficult for a TTS system to detect it automatically.

14.3.6. Web Pages

All the comments about TTS reliance on XML markup of document structure can be applied to the case of HTML-marked Web page content as well. In addition to sections, headers, lists, paragraphs, etc., the TTS systems should be aware of XML/HTML conventions such as links (<a href="…">link name</a>) and perhaps apply some distinctive voice quality or prosodic pitch contour to highlight these. The size and color of the section of text also provides useful hints for emphasis. Moreover, the TTS system should also integrate the rendering of audio and video contents on the Web page to create a genuine multimedia experience for the users. More could be said about the rendition of Web content, whether from underlying XML documents or HTML-marked documents prepared specifically for Web presentation. In addition, the World Wide Web Consortium has begun work on standards for aural stylesheets that can work in conjunction with standard HTML to provide special direction in aural rendition [33].

14.3.7. Dialog Turns and Speech Acts

Not all text to be rendered by a TTS system is standard written prose. The more expressive TTS systems could be tasked with rendering natural conversation and dialog in a spontaneous style. As with written documents, the TTS system has to be guided by XML markup of its input. Various systems for marking *dialog turns* (change of speaker) and *speech acts* (the mood and functional intent of an utterance)[3] are used for this purpose, and these annotations will trigger particular phonetic and prosodic rules in future TTS systems. The speech act coding schemes can help, for example, in identifying the speaker's intent with respect to an utterance, as opposed to the utterance's structural attributes. The prosodic contour and voice quality selected by the TTS system might be highly dependent on this functional knowledge.

For example, a syntactically well-formed question might be used as information solicitation, with the typical utterance-final pitch upturn as shown in the following:

<REQUEST_INFO>Can you hand me the wrench?</REQUEST_INFO>

But if the same utterance is used as a command, the prosody may change drastically.

<DIRECTIVE>Can you hand me the wrench.</DIRECTIVE>

Research on speech act markup-tag inventories (see Chapter 17) and automatic methods for speech act annotation of dialog is ongoing, and this research has the property considered desirable here, in that it is independently motivated (useful for enhancing speech recognition and language understanding systems). Thus, an advanced TTS system should be expected to exploit dialog and speech act markups extensively.

[3] Dialog modeling and the concepts of *dialog turns* and *speech acts* are described in detail in Chapter 17.

14.4. TEXT NORMALIZATION

Text often include abbreviations (e.g., FDA for *Food and Drug Administration*) and acronyms (SWAT for *Special Weapons And Tactics*). Novels and short stories may include *spoken* dialog interspersed with exposition; technical manuals may include mathematical formulae, graphs, figures, charts and tables, with associated captions and numbers; e-mail may require interpretation of special conventional symbols such as *emoticons* [e.g., :-) means smileys], as well as Web and interInternet address formats, and special abbreviations (e.g., IMHO means *in my humble opinion*). Again, any text source may include part numbers, stock quotes, dates, times, money and currency, and mathematical expressions, as well as standard ordinal and cardinal formats. Without context analysis or prior knowledge, even a human reader would sometimes be hard pressed to give a perfect rendition of every sequence of nonalphabetic characters or of every abbreviation. *Text normalization* (TN) is the process of generating normalized orthography (or, for some systems, direct generation of phones) from text containing words, numbers, punctuation, and other symbols. For example, a simple example is given as follows:

The 7% Solution ➔ THE SEVEN PER CENT SOLUTION

Text normalization is an essential requirement not only for TTS, but also for the preparation of training text corpora for acoustic-model and language-model construction.[4] In addition, speech dictation systems face an analogous problem of *inverse* text normalization for document creation from recognized words, and such systems may depend on knowledge sources similar to those described in this section. The example of an inverse text normalization for the example above is given as follows:

THE SEVEN PER CENT SOLUTION ➔ The 7% Solution

Modular text normalization components, which may produce output for multiple downstream consumers, mark up the exemplary text along the following lines:

The <tn snor=”SEVEN PER CENT”>7%</tn> Solution

The `snor` tag stands for *Standard Normalized Orthographic Representation.*[5] For TTS, input text may include multisentence paragraphs, numbers, dates, times, punctuation, symbols of all kinds, as well as interpretive annotations in a TTS markup language, such as tags for word emphasis or pitch range. Text analysis for TTS is the work of converting such text into a stream of normalized orthography, with all relevant input tagging preserved and new markup added to guide the subsequent modules. Such interpretive annotations added by text analysis are critical for phonetic and prosodic generation phases to produce desired output. The output of the text normalizer may be deterministic, or may preserve a full set of interpretations and processing history with or without probabilistic information to be passed along to later stages. We once again assume that XML markup is an appropriate format for

[4] For details of acoustic and language modeling, please refer to Chapters 9 and 11.

[5] SNOR, or Standard Normalized Orthographic Representation, is a uniform way of writing words and sentences that corresponds to spoken rendition. SNOR-format sentence texts are required as reference material for many Defense Advanced Research Project Agency and National Institutes of Standards and Technology-sponsored standard speech technology evaluation procedures.

expressing knowledge that can be created by a variety of external processes and exploited by a number of technologies in addition to TTS.

Since today's TTS systems typically cannot expect that their input be independently marked up for text normalization, they incorporate internal technology to perform this function. Future systems may piggyback on full natural language processing solutions developed for independent purposes. Presently, many incorporate minimal, TTS-specific hand-written rules [1], while others are loose agglomerations of modular, task-specific statistical evaluators [3].

For some purposes, an architecture that allows for a set or lattice of possible alternative expansions may be preferable to deterministic text normalization, like the *n-best* lists or *word graph* offered by the speech recognizers described in Chapter 13. Alternatives known to the system can be listed and ranked by probabilities that may be learnable from data. Later stages of processing (linguistic analysis or speech synthesis) can either add knowledge to the lattice structure or recover the best alternative, if needed. Consider the fragment "*at 8 am I . . .* " in some informal writing such as e-mail. Given the flexibility of writing conventions for pronunciation, *am* could be realized as either *A. M.* (the numeric context seems to cue at times) or the auxiliary verb *am*. Both alternatives could be noted in a descriptive lattice of covering interpretations, with confidence measures if known.

Table 14.2 Two alternative interpretations for sentence fragment "*At 8 am I* …".

At 8 am I …	At <time> eight am </time> I …
At 8 am I …	At <number> eight </number> am I …

If the potential ambiguity in the interpretation of *am* in the above pair of examples is simply noted, and the alternatives retained rather than suppressed, the choice can be made by a later stage of syntactic/semantic processing. Note another feature of this example—the rough irregular abbreviation form for antemeridian, which by prescriptive convention hopes that high-quality TTS processing can rely entirely on *standard* stylistic conventions. That observation also applies to the *obligatory* use of "?" for all questions.

Specific architectures for the text normalization component of TTS may be highly variable, depending on the system architect's answers to the following questions:

- Are cross-functional language processing resources mandated, or available?
- If so, are phonetic forms, with stress or accent, and normalized orthography, available?
- Is a *full* syntactic and semantic analysis of input text mandated, or available?
- Can the presenting application add interpretive knowledge to structure the input (text)?
- Are there interface or pipelining requirements that preclude lattice alternatives at every stage?

Because of this variability in requirements and resources, we do not attempt to formally specify a single, all-purpose architectural solution here. Rather, we concentrate on describing the text normalization challenges any system has to face. We note where solutions to these challenges are more readily realized under particular architectural assumptions.

All text normalization consists of two phases: identification of type, and expansion to SNOR or other unambiguous representation. Much of the identification phase, dealing with phenomena of sentence boundary determination, abbreviation expansion, number spell-out, etc. can be modeled as regular processes (see Chapter 11). This raises an interesting architectural issue. Imagine a system based entirely on regular *finite state transducers* (FST, see Chapter 11), as in [27], which enforces an appealing uniformity of processing mechanism and internal structure description. The FST permits a lattice-style representation that does not require premature resolution of any structural choice. An entire text analysis system can be based on such a representation. However, as long as a system confines its attention to issues that commonly come under the heading of text normalization, such as number formats, abbreviations, and sentence breaking, a simpler regular-expression-based uniform mechanism for rule specification and structure representation may be adequate.

Alternatively, TTS systems could make use of advanced tools such as, for example, the lex and yacc tools [17], which provide frameworks for writing customized lexical analyzers and context-free grammar parsers, respectively. In the discussion of typical text normalization requirements below, examples will be provided and then a fragment of Perl pattern-matching code will be shown that allows matching of the examples given. Perl notation [36] is used as a convenient short-hand representing any equivalent regular expression parsing system and can be regarded as a subset of the functionality provided by any regular expression, FST, or context-free grammar tool set that a TTS software architect may choose to employ. Only a small subset of the simple, fairly standard Perl conventions for regular expression matching are used, and comments are provided in our discussion of text normalization.

A text normalization system typically adds identification information to assist subsequent stages in their tasks. For example, if the TN subsystem has determined with some confidence that a given digit string is a phone number, it can associate XML-like tags with its output, identifying the corresponding normalized orthographic chunk as a candidate for special phone-number intonation. In addition, the identification tags can guide the lexical disambiguation of terms for other processes, like phonetic analysis in TTS systems and training data preparation for speech recognition.

Table 14.3 shows some examples of input fragments with a relaxed form of output normalized orthography. It illustrates a possible ambiguity in TN output. In the (contrived) example, the ambiguity is between a place name and a hypothetical individual named perhaps *Steve* or *Samuel* Asia. Two questions arise in such cases. The first is format of specification. The data between submodules in a TTS system can be passed (or be placed in a centrally viewable *blackboard* location) as tagged text or in a binary format. This is an implementation detail. Most important is that all possibilities known to the TN system be specified in the output, and that confidence measures from the TN, if any, be represented. For example, in many contexts, *South Asia* is the more likely spell-out of *S. Africa*, and this should be indicated implicitly by ordering output strings, or explicitly with probability numbers. The decision could then be delayed until one has enough information in the later module (like linguistic analysis) to make the decision in an informed manner.

Table 14.3 Examples of the normalized output using XML-like tags for text normalization.

Dr. King	<title> DOCTOR </title> KING
7%	<number>SEVEN<ratio>PERCENT</ratio> </number>
S. Asia	<toponym> SOUTH ASIA </toponym> OR <psn_name><initial>S</initial>ASIA</psn_name>

14.4.1. Abbreviations and Acronyms

As noted above, a period is an important but not completely reliable clue to the presence of an abbreviation. Periods may be omitted or misplaced in text for a variety of reasons. For similar reasons of stylistic variability and a writer's (lack of) care and skill, capitalization, another potentially important clue, can be variable as well. For example, all the representations of the abbreviation for *post script* listed below have been observed in actual mail and e-mail. A system must therefore combine knowledge from a variety of contextual sources, such as document structure and origin, when resolving abbreviations:

PS. Don't forget your hat.
Ps. Don't forget your hat.
P.S. Don't forget your hat.
P.s. Don't forget your hat.

And *P.S.*, when examined out of context, could be personal name initials as well. Of course, a given TTS system's user may be satisfied with the simple spoken output */p iy ae s/* in cases such as the above, obviating the need for full interpretation. But at a minimum, when *fallback to letter pronunciation* is chosen, the TTS system must attempt to ensure that some obvious spell-out is not being overlooked. For example, a system should not render the title in *Dr. Jones* as letter names */d iy aa r/*.

Actually, any abbreviation is potentially ambiguous, and there are several distinct types of ambiguity. For example, there are abbreviations, typically quantity and measure terms, which can be realized in English as either plural or singular depending on their numeric coefficient, such as *mm* for *millimeter(s).* This type of ambiguity can get especially tricky in the context of conventionally frozen items. For example, *9mm ammunition* is typically spoken as *nine millimeter ammunition* rather than *nine millimeters ammunition*.

Next, there are forms that can, with appropriate syntactic context, be interpreted either as abbreviations or as simple English words, such as *in* (inches), particularly at the end of sentences.

Finally, many, perhaps most, abbreviations have entirely different abbreviation spell-outs depending on semantic context, such as *DC* for *direct current* or *District of Columbia*. This variability makes it unlikely that any system ever performs perfectly. However, with sufficient training data, some statistical guidelines for interpretation of common abbreviations in context can be derived. Table 14.4 shows a few more examples of this most difficult type of ambiguity:

Table 14.4 Some ambiguous abbreviations.

CO	Colorado	commanding officer
	conscientious objector	carbon monoxide
IRA	Individual Retirement Account	Irish Republican Army
MD	Maryland	doctor of medicine
	muscular dystrophy	

An advanced TTS system should attempt to convert reliably at least the following abbreviations:

- Title—Dr., MD, Mr., Mrs, Ms., St. (Saint), … etc.
- Measure—ft., in., mm, cm (centimeter), kg (kilogram), … etc.
- Place names—CO, LA, CA, DC, USA, st. (street), Dr. (Drive), … etc.

Abbreviation disambiguation usually can be resolved by POS (part-of-speech) analysis. For example, whether *Dr*. is *Doctor* or *Drive* can be resolved by examining the POS features of the previous and following words. If the abbreviation is followed by a capitalized personal name, it can be expanded as *Doctor*, whereas if the abbreviation is preceded by a capitalized place name, a number, or an alphanumeric (like 120th), it will be expanded as *Drive*. Although the example above is resolved via a series of heuristic rules, the disambiguation (POS analysis) can also be done by a statistical approach. In [6], the POS tags are determined based on the most likely POS sequence using POS trigram and lexical-POS unigram. Since an abbreviation can often be distinguished by its POS feature, the most likely POS sequence of the sentence discovered by the trigram search then provides the best guess of the POS (thus the usage) for abbreviations [6]. We describe POS tagging in more detail in Section 14.5.

Other than POS information, the lexical entries for abbreviations should include all features and alternatives necessary to generate a lattice of possible analyses. For example, a typical abbreviation's entry might include information as to whether it could be a word (like *in*), whether period(s) are optional or required, whether plural variants must be generated and if so under what circumstances, whether numerical specification is expected or required, etc.

Acronyms are words created from the first letters or parts of other words. For example, SCUBA is an acronym for *self-contained underwater breathing apparatus*. Generally to qualify as a true acronym, a letter sequence should reflect normal language phonotactics, such as a reasonable alternation of consonants and vowels. From a TTS system's point of view, the distinctions between acronyms, abbreviations, and plain new or unknown words can be unclear. Many acronyms can be entered into the TTS system lexicon just as ordinary words would be. However, unknown acronyms (not listed in the lexicon) may occasionally be encountered. Although an acronym's case property can be a significant clue to identification, it is often unclear how to speak a given sequence of upper-case letters. Most TTS systems, failing to locate the sequence in the acronym dictionary, spell it out letter-by-letter. Other systems attempt to determine whether the sequence is inherently *speakable*. For example, DEC might be inherently speakable, while FCC is not formed according to normal word phonotactics. When something speakable is found, it is processed via the normal letter-to-sound rules, while something *unspeakable* would be spelled out letter-by-letter. Yet other systems might simply feed the sequence directly to the letter-to-sound rules (see Section 14.8), just as they would any other unknown word. As with all such problems, a larger lexicon usually provides superior results.

ALGORITHM 14.2 ABBREVIATIONS AND ACRONYMS EXPANSION

```
1.  If word token w is not in abbreviation table goto 3.
2.  Abbreviation Expansion
    If the POS tag of w and the correspondent abbreviation match
      Abbreviation expansion by inserting SNOR and interpretive annotation tags
      Advance one word and goto 1.
3.  Acronym Expansion
     If w contains only capital letters
     If w is in the predefined acronym table
      Acronym expansion by inserting SNOR and interpretive annotation tags
      according to acronym expansion table
     else spell out w letter-by-letter
4.  Advance one word and goto 1.
```

The general algorithm for abbreviations and acronyms expansion in text normalization is summarized in Algorithm 14.2. The algorithm assumes that tokenization and POS tagging have been done for the whole sentence. Abbreviation expansion is determined by the POS tags of the potential abbreviation candidates. Acronym expansion is done exclusively by table lookup, and letter-by-letter spell-out is used when acronyms cannot be found in the acronym table.

14.4.2. Number Formats

Numbers occur in a wide variety of formats and have a wide variety of contextually dependent reading styles. For example, the digits 370 in the context of the product name *IBM 370 mainframe computer* typically are read as *three seventy*, while in other contexts 370 would be read as *three hundred seventy* or *three hundred and seventy*. In a phone number, such as 370-1111, the string would normally be read as *three seven oh*, while in still other contexts it might be rendered as *three seven zero*. A text analysis system can incorporate rules, perhaps augmented by probabilities, for these situations, but might never achieve perfection in all cases. Phone numbers are a practical place to start, and their treatment illustrates some of the general issues relevant to the other number formats which are covered below.

14.4.2.1. Phone Numbers

Phone numbers may include prefixes and area codes and may have dashes and parentheses as separators. Examples are shown in Table 14.5.

Table 14.5 Some different written representations of phone numbers.

9-999-4118
9 345-5555
(617) 932-9209
(617) 932-9209
716-123-4568
409/845-2274
+49 (228) 550-381
+49-228-550-381

The first two examples have *prefix* codes, while the next three have area codes with minor formatting differences. The final two examples are possible international-format phone numbers. A basic Perl regular expression pattern to subsume the commonality in all the local domestic numbers can be defined as follows:

```
$us_basic = '([0-9]{3}\-[0-9]{4})';
```

This defines a pattern subpart to match 3 digits, followed by a separator dash, followed by another 4 digits. Then the pattern to match the prefix type would be:

```
/([0-9]{1})[\/ -]($us_basic)/
```

In the example above, this leaves the system pattern variable $1 (corresponding to the first set of capture parentheses in the pattern) set to 1, and $2 (the second set of capture parentheses) set to 999-4118. Then a separate set of tables, indexed by the rule name and the pattern variable contents, could provide orthographic spell-outs for the digits. Clearly a balance has to be struck between the number of pattern variables provided in the expression and the overall complexity of the expression, vis-à-vis the complexity and sophistication of the indexing scheme of the spell-out tables. For example, the $us_basic could be defined to incorporate parentheses capture on the first three digits and the remaining four separately, which might lead to a simpler spell-out table in some cases.

The pattern to match the area code types could be:

```
/(\([0-9]{3}\))[\/ -]($us_basic)/
```

These patterns could be endlessly refined, expanded, and layered to match strings of almost arbitrary complexity. A balance has to be struck between number and complexity of distinct patterns. In any case, no matter how sophisticated the matching mechanism, arbitrary or at best probabilistic decisions have to be made in constructing a TTS system. For example, in matching an area code type, the rule architect must decide how much and what kind of whitespace separation the matching system tolerates between the area code and the rest of the number before a phone-number match is considered unlikely. Or, as another example, does the rule architect allow new lines or other formatting characters to appear between the area code and the basic phone number? These kinds of decisions must be explicitly considered, or made by default, and should be specified to a reasonable degree in user documentation. There are a great many other phone number formats and issues that are beyond the scope of this treatment.

Once a certain type of pattern requires a conversion to normalized orthography, the question of how to perform the conversion arises. The conversion characters can be aligned with the identification, so that conversion occurs implicitly during the pattern matching process. Another way is to separate the conversion from the identification phase. This may or may not lead to gains in efficiency and elimination of redundancy, depending on the overall architecture of the system and whether and how components are expected to be reused. A version of this second approach is sketched here.

Suppose that the pattern match variable $1 has been set to 617 by one of the identification-phase pattern matches described above. Another list can provide pointers to conversion tables, indexed by the rule name or number and the variable name. So for the rule that can match area codes formatted as in (c) above, the relevant entry would be:

Identification rule name	Variable	Spellout table name
Area-Phone	$1	LITERAL_DIGIT

The LITERAL_DIGIT spell-out rule set, when presented with the 617 character sequence (the value of $1), simply generates the normalized orthography *six one seven*, by table lookup. In this simple and straightforward approach, spell-out tables such as LITERAL_DIGIT can be reused for portions of a wide variety of identification rules. Other simple numeric spell-out tables would cover different styles of numeric reading, such as *pairwise* style (e.g., *six seventeen*), full decimal with tens, hundreds, thousands units (*six hundred seventeen*), and so on. Some spellout tables may require processing code to supplement the basic table lookup. Additional examples of spell-out tables are not provided for the various other types of text normalization entities exemplified below, but would function similarly.

14.4.2.2. Dates

Dates may be specified in a wide variety of formats, sometimes with a mixture of orthographic and numeric forms. Note that dates in TTS suffer from a mild form of the century-date-change uncertainty (the infamous Y2K bug), so a form such as 5/7/37 may in the future be ambiguous, in its full form, between 1937 and 2037. The safest course is to say as little as possible, i.e., "*five seven thirty seven*", or even "*May seventh, thirty seven*", rather than attempt "*May seventh, nineteen thirty seven*". Table 14.6 shows a variety of date formats and associated normalized orthography.

Table 14.6 Various date formats.

12/19/94 (US)	December nineteenth ninety four
19/12/94 (European)	December nineteenth ninety four
04/27/1992	April twenty seventh nineteen ninety two
May 27, 1995	May twenty seventh nineteen ninety five
July 4, 94	July fourth ninety four
1,994	one thousand nine hundred and ninety four
1994	nineteen ninety four

One issue that comes up with certain number formats, including dates, is range checking. A form like 13/19/94 is basically uninterpretable as a date. This kind of checking, if included in the initial pattern matching, may be slow and may increase formal requirements for power of the pattern matching system. Therefore, range checking can be done at spell-out time (see below) during normalized orthography generation, as long as a backtracking or redo option is present. If range checking is desired as part of the basic identification phase of text normalization, some regular expression matching systems allow for extensions. For example, the following pattern variable matches only numbers less than or equal to 12, the valid month specifications. It can be included as part of a larger, more complex date matching *pattern:*

```
$month = '/(0[123456789]/1[012]/'
```

14.4.2.3. Times

Times may include hours, minute, seconds, and duration specifications as shown in Table 14.7. Time formats exemplify yet another area where linguistic concerns have to intersect with architecture. If simple, flat normalized orthography is generated during a text normalization phase, a later stage may still find a form like *am* ambiguous in pronunciation. If a lattice of alternative interpretations is provided, it should be supplemented with interpretive information on the linguistic status of the alternative text analyses. Alternatively, a single best guess can be made, but even in this case, some kind of interpretive information indicating the status of the choice as, e.g., a time expression should be provided for later stages of syntactic, semantic, and prosodic interpretation. This reiterates the importance of TTS text analysis systems to generate interpretive annotations tags for subsequent modules' use whenever possible, as discussed in Section 14.4. In some cases, unique text formatting of the choice, corresponding to the system's lexical contents, may be sufficient. That is, in some systems, generation of *A.M.*, for example, may uniquely correspond to the lexicon's entry for that portion of a time expression, which specifies the desired pronunciation and grammatical treatment.

Table 14.7 Several examples for time expressions.

11:15	eleven fifteen
8:30 pm	eight thirty pm
5:20 am	five twenty am
12:15:20	twelve hours fifteen minutes and twenty seconds
07:55:46	seven hours fifty-five minutes and forty-six seconds

14.4.2.4. Money and Currency

As illustrated in Table 14.8, money and currency processing should correctly handle at least the currency indications $, £, DM, ¥, and €, standing for dollars, British pounds, Deutsche marks, Japanese yen, and euros, respectively. In general, $ and £ have to precede the numeral; DM, ¥, and € have to follow the numeral. Other currencies are often written in full words and have to follow the numeral, though abbreviations for these are sometimes found, such as *100 francs* and *20 lira*.

Table 14.8 Several money and currency expressions.

$40	FORTY DOLLARS
£200	TWO HUNDRED POUNDS
5 ¥	FIVE YEN
25 DM	TWENTY FIVE DEUTSCH MARKS
300€	THREE HUNDRED EUROS

14.4.2.5. Account Numbers

Account numbers may refer to bank accounts or social security numbers. Commercial product part numbers often have these kinds of formats as well. In some cases these cannot be readily distinguished from mathematical expressions or even phone numbers. Some examples are shown below:

```
123456-987-125456
000-1254887-87
049-85-5489
```

The other popular number format is that of credit card number, such as

```
4446-2289-2465-7065
3745-122267-22465
```

To process formats like these, it may eventually be desirable for TTS systems to provide customization capabilities analogous to the pronunciation customization features for words found in current TTS systems. Regular expression formalisms of the type exemplified above for phone number, would, if exposed to applications and developers through suitable editors, be adequate for most such needs.

14.4.2.6. Ordinal Numbers

Ordinal numbers are those referring to rank or placement in a series. Examples include:

1st, 2nd, 3rd, 4th, 10th, 11th , 12th, 20th, 100th, 1000th, etc.
1th, 2nd, 3rd, 4th, 10th, 11th, 12th, 20th, 21st, 32nd, 100th, 1000th, etc.

The system's ordinal processing may also be used to generate the denominators of fractions, except for halves, as shown in Table 14.9. Notice that the ordinal must be plural for numerators other than 1.

Table 14.9 Some examples of fractions.

1/2	one half
1/3	one third
1/4	one quarter or one fourth
1/10	one tenth
3/10	three tenths

14.4.2.7. Cardinal Numbers

Cardinal numbers are, loosely speaking, those forms used in simple counting or the statement of amounts. If a given sequence of digits fails to fit any of the more complex formats above, it may be a simple cardinal number. These may be explicitly negative or positive or assumed positive. They may include decimal or fractional specifications. They may be read in several different styles, depending on context and/or aesthetic preferences. Table 14.10 gives some examples of cardinal numbers and alternatives for normalized orthography.

Table 14.10 Some cardinal number types.

123	one two three	one hundred (and) twenty three
1,230	one thousand two hundred (and) thirty	
2426	two four two six	twenty four twenty six
	two thousand (and) twenty six	

The number-expansion algorithm is summarized in Algorithm 14.3. In this algorithm the text normalization module maintains an extensive pattern table. Each pattern in the table contains its associated pattern in regular expression or Perl format along with a pointer to a rule in the conversion table, which guides the expansion process.

A regular expression to match well-formed cardinals with commas grouping chunks of three digits of the type from 1,000,000 to 999,999,999 might appear as:

```
if ($item =~ /^([0-9]{1,3}),([0-9]{3}),([0-9]{3})/
 {        $NewFrame->{"millions"} = $1;
    $NewFrame->{"thousands"} = $2;
    $NewFrame->{"hundreds"} = $3;
     print "Grouped cardinal found: $item\n";
    return $NewFrame;       }
```

ALGORITHM 14.3 NUMBER EXPANSION

```
1.  Pattern Matching
      If a match is found goto 2.
      else goto 3.
2.  Number Expansion
      Insert SNOR and interpretive annotation tags according to the associated rule
      Advance the pointer to the right of the match pattern and goto 1.
3.  Finish
```

14.4.3. Domain-Specific Tags

In keeping with the theme of this section—that is, the increasing importance of independently generated precise markup of text entities—we present a little-used but interesting example.

14.4.3.1. Mathematical Expressions

Mathematical expressions are regarded by some systems as the domain of special-purpose processors. It is a serious question how far to go in mathematical expression parsing, since providing some capability in this area may raise users' expectations to an unrealistic level. The World Wide Web Consortium has developed MathML (mathematical markup language) [34], which provides a standard way of describing math expressions. MathML is an XML extension for describing mathematical expression structure and content to enable mathematics to be served, received, and processed on the Web, similar to the function HTML has performed for text. As XML becomes increasingly pervasive, MathML could possibly be used to guide interpretation of mathematical expressions. For the notation $(x + 2)^2$ a possible MathML representation such as that below might serve as an initial guide for a spoken rendition.

```
<EXPR>
 <EXPR>
 x
```

```
   <PLUS/>
   2
   </EXPR>
   <POWER/>
   2
  </EXPR>
```

This might be generated by an application or by a specialized preprocessor within the TTS system itself. Prosodic rules or data tables appropriate for math expressions could then be triggered.

14.4.3.2. Chemical Formulae

As XML becomes increasingly common and exploitable by TTS text normalization, other areas follow. For example, Chemical Markup Language (CML [22]) now provides a standard way to describe molecular structure or chemical formulae. CML is an example of how standard conventions for text markup are expected increasingly to replace ad hoc, TTS-internal heuristics.

In CML, the chemical formula C_2OCOH_4 would appear as:

```
<FORMULA>
    <XVAR BUILTIN="STOICH">
    C C O C O H H H H
    </XVAR>
</FORMULA>
```

It seems reasonable to expect that TTS engines of the future will be increasingly devoted to interpreting such precise conventions in high-quality speech renditions rather than endlessly replicating NL heuristics that fail as often as they succeed in guessing the identity of raw text strings.

14.4.4. Miscellaneous Formats

A random list illustrating the range of other types of phenomena for which an English-oriented TTS text analysis module must generate normalized orthography might include:

- Approximately/tilde: The symbol ~ is spoken as *approximately* before (Arabic) numeral or currency amount, otherwise it is the character named *tilde*.
- Folding of accented Roman characters to *nearest* plain version: If the TTS system has no knowledge of dealing with foreign languages, like French or German, a table of folding characters can be provided so that for a term such as *Über-mensch*, rather than spell out the word *Über*, or ignore it, the system can convert it to its *nearest* English-orthography equivalent: *Uber*. The ultimate way to process such foreign words should integrate a language identification module

with a multi-lingual TTS system, so that language-specific knowledge can be utilized to produce appropriate text normalization of all text.

- Rather than simply ignore high ASCII characters in English (characters from 128 to 255), the text analysis lexicon can incorporate a table that gives *character names* to all the printable high ASCII characters. These names are either the full Unicode character names, or an abbreviated form of the Unicode names. This would allow speaking the names of characters like © (*copyright sign*), ™ (*trademark*), @ (*at*), ® (*registered mark*), and so on.
- Asterisk: in email, the symbol '*' may be used for emphasis and for setting off an item for special attention. The text analysis module can introduce a little pause to indicate possible emphasis when this situation is detected. For the example of "*Larry has *never* been here*," this may be suppressed for asterisks spanning two or more words. In some texts, a word or phrase appearing completely in UPPER CASE may also be a signal for special emphasis.
- Emoticons: There are several possible emoticons (emotion icons).

1.	:-) or :)	SMILEY FACE (humor, laughter, friendliness, sarcasm)
2.	:-(or :(	FROWNING FACE (sadness, anger, or disapproval)
3.	;-) or ;)	WINKING SMILEY FACE (naughty)
4.	:-D	OPEN-MOUTHED SMILEY FACE (laughing out loud)

Smileys, of which there are dozens of types, may be tacked onto word start or word end or even occur interword without spaces, as in the following examples.

:)hi!
Hi:)
Hi:)Hi!

14.5. LINGUISTIC ANALYSIS

Linguistic analysis (sometimes also referred to as syntactic and semantic parsing) of natural language (NL) constitutes a major independent research field. Often commercial TTS systems incorporate some minimal parsing heuristics developed strictly for TTS. Alternatively, the TTS systems can also take advantage of independently motivated natural language processing (NLP) systems, which can produce structural and semantic information about sentences. Such linguistically analyzed documents can be used for many purposes other than TTS—information retrieval, machine translation system training, etc.

Provision of some parsing capability is useful to TTS systems in several areas. Parsers may be used in disambiguating the text normalization alternatives described above. Additionally, syntactic/semantic analysis can help to resolve grammatical features of individual words that may vary in pronunciation according to sense or abstract inflection, such as *read*.

Finally, parsing can lay a foundation for derivation of a prosodic structure useful in determining segmental duration and pitch contour.

The fundamental types of information desired for TTS from a parsing analysis are summarized below:

- Word part of speech (POS) or word type, e.g., proper name or verb.
- Word sense, e.g., *river bank* vs. *money bank*.
- Phrasal cohesion of words, such as idioms, syntactic phrases, clauses, sentences.
- Modification relations among words.
- Anaphora (co-reference) and synonymy among words and phrases.
- Syntactic type identification, such as questions, quotes, commands, etc.
- Semantic focus identification (emphasis).
- Semantic type and speech act identification, such as requesting, informing, narrating, etc.
- Genre and style analysis.

Here we confine ourselves to discussion of the kind of information that a good parser could, in principle, provide to enable the TTS-specific functionality.

Linguistic analysis supports the phonetic analysis and prosodic generation phases. The modules of phonetic analysis are covered in Sections 14.6, 14.7, and 14.8. A linguistic parser can contribute in several ways to the process of generating (symbolic) phonetic forms from orthographic words found in text. One function of a parser is to provide accurate part-of-speech (POS) labels. This can aid in resolving the pronunciation of several hundred American English homographs, such as *object* and *absent*. Homographs are discussed in greater detail in Section 14.6 below. Parsers can also aid in identifying names and other special classes of vocabulary for which specialized pronunciation rule sets may exist [32].

Prosody generation deals mainly with assignment of segmental duration and pitch contour that have close relationship with prosodic phrasing (pause placement) and accentuation. Parsing can contribute useful information, such as the syntactic type of an utterance. (e.g., yes/no question contours typically differ from *wh*-question contours, though both are marked simply by '?' in text), as well as semantic relations of synonymy, anaphora, and focus that may affect accentuation and prosodic phrasing. Information from discourse analysis and text genre characterization may affect pitch range and voice quality settings. Further examination of the contribution of parsing specifically to prosodic phrasing, accentuation, and other prosodic interpretation is provided in Chapter 15.

As mentioned earlier, TTS can employ either a general-purpose NL analysis engine or a pipeline of a number of very narrowly targeted, special-purpose NL modules together for the requirement of TTS linguistic analysis. Although we focus on linguistic information for supporting phonetic analysis and prosody generation here, a lot of the information and services are beneficial to document structure detection and text normalization described in previous sections.

The minimum requirement for such a linguistic analysis module is to include a lexicon of the *closed-class* function words, of which only several hundred exist in English (at most), and perhaps homographs. In addition, a minimal set of modular functions or services would include:

- *Sentence breaking*—Sentence breaking has been discussed in Section 14.3.4 above.
- *POS tagging*—POS tagging can be regarded as a two-stage process. The first is POS guessing, which is the process of determining, through a combination of a (possibly small) dictionary and some morphological heuristics or a specialized morphological parser, the POS categories that might be appropriate for a given input term in isolation. The second is POS choosing—that is, the resolution of the POS in context, via local short-window syntactic rules, perhaps combined with probabilistic distribution for the POS guesses of a given word. Sometimes the guessing and choosing functions are combined in a single statistical framework. In [6], lexical probabilities are unigram frequencies of assignments of categories to words estimated from corpora. For example, in the original formulation of the model, the lexical probabilities [$P(c_i \mid w_i)$, where c_i is the hypothesized POS for word w_i], were estimated from the hand-tagged Brown corpus [8], and the word *see* appeared 771 times as a verb and once as an interjection. Thus the probability that *see* is a verb is estimated to be 771/772 or 0.99. Trigrams are used for contextual probability [$P(c_i \mid c_{i-1}c_{i-2}\cdots c_1) = P(c_i \mid c_{i-1}c_{i-2})$]. Lexical probabilities and trigrams over category sequences are used to score all possible assignments of categories to words for a given input word sequence. The entire set of possible assignments of categories to words in sequence is calculated ,and the best-scoring sequence is used. Likewise, simple methods have been used to detect noun phrases (NPs), which can be useful in assigning pronunciation, stress, and prosody. The method described in [6] relies on a table of probabilities for inserting an NP begin bracket '[' between any two POS categories, and similarly for an NP end bracket ']'. This was also trained on the POS-labeled Brown corpus, with further augmentation for the NP labels. For example, the probability of inserting an NP begin bracket after an article was found to be much lower than that of begin-bracket insertion between a verb and a noun, thus automatically replicating human intuition.
- *Homograph disambiguation*—Homograph disambiguation in general refers to the case of words with the same orthographic representation (written form) but having different semantic meanings and sometimes even different pronunciations. Sometimes it is also referred as sense disambiguation. Examples include "The boy used the *bat* to hit a home run" vs. "We saw a large *bat* in the zoo" (the pronunciation is the same for two *bat*) and "You *record* your voice" vs. "I'd like to buy that *record*" (the pronunciations are different for the two *record*). The linguistic analysis module should at least try to resolve the ambiguity for the case of different pronunciations because it is absolutely required for correct phonetic rendering. Typically, the ambiguity can be resolved based on POS and

lexical features. Homograph disambiguation is described in detail in Section 14.6.

- *Noun phrase (NP) and clause detection*—Basic NP and clause information could be critical for a prosodic generation module to generate segmental durations. It also provides useful cues to introduce necessary pauses for intelligibility and naturalness. Phrase and clause structure are well covered in any parsing techniques.
- Sentence type identification—Sentence types (declarative, yes-no question, etc.) are critical for macro-level prosody for the sentence. Typical techniques for identifying sentence types have been covered in Section 14.3.4.

If a more sophisticated parser is available, a richer analysis can be derived. A so-called *shallow parse* is one that shows syntactic bracketing and phrase type, based on the POS of words contained in the phrases. A training corpus of shallow-parsed sentences has been created for the Linguistic Data Consortium [16]. The following example illustrates a shallow parse for sentence : "For six years, Marshall Hahn Jr. has made corporate acquisitions in the George Bush mode: kind and gentle."

```
For/IN[six/CD years/NNS],/,[T./NNP Marshall/NNP
Hahn/NNP Jr./NNP]has/VBZ made/VBN[corporate/JJ acquisi-
tions/NNS]in/IN[the/DT George/NNP Bush/NNP mode/NN]
:/:[kind/JJ]and/CC[gentle/JJ]./.
```

The POS labels used in this example are described in Chapter 2 (Table 2.14). A TTS system uses the POS labels in the parse to decide alternative pronunciations and to assign differing degrees of prosodic prominence. Additionally, the bracketing might assist in deciding where to place pauses for great intelligibility. A fuller parse would incorporate more higher-order structure, including sentence type identification, and more semantic analysis, including co-reference.

14.6. HOMOGRAPH DISAMBIGUATION

For written languages, *sense* ambiguities occur when words have different syntactic/semantic meanings. Those words with different senses are called *polysemous* words. For example, *bat* could mean either a kind of animal or the equipment to hit a baseball. Since the pronunciations for the two different senses of *bat* are identical, we are in general only concerned[6] about the other type of polysemous words that are homographs (spelled alike but vary in pronunciation), such as *bass* for a kind of fish (/*b ae s*/) or an instrument (/*b ey s*/).

Homograph variation can often be resolved on POS (grammatical) category. Examples include *object*, *minute*, *bow*, *bass*, *absent*, etc. Unfortunately, correct determination of POS (whether by a parsing system or statistical methods) is not always sufficient to resolve pro-

[6] Sometimes, a polysemous word with the same pronunciation could have impact for prosodic generation because different semantic properties could have different accentuation effects. Therefore, a high-quality TTS system can definitely be benefited from word-sense disambiguation beyond homograph disambiguation.

nunciation alternatives. For example, simply knowing that the form *bow* is a noun does not allow us to distinguish the pronunciation appropriate for the instrument of archery from that for the front part of a boat. Even more subtle is the pronunciation of *read* in “If you *read* the book, he’ll be angry.” Without contextual clues, even human readers cannot resolve the pronunciation of *read* from the given sentence alone. Even though the past tense is more likely in some sense, deep semantic and/or discourse analysis would be required to resolve the tense ambiguity.

Several hundred English homographs extracted from the 1974 *Oxford Advanced Learners Dictionary* are listed in [10]. Here are some examples:

- Stress homographs: noun with front-stress vowel, verb with end-stress vowel
  “an absent *boy” vs. “Do you choose to* absent *yourself?”*
- Voicing: noun/verb or adjective/verb distinction made by voice final consonant
  “They will abuse *him.” vs. “They won’t take* abuse*.”*
- –ate words: noun/adjective sense uses schwa, verb sense uses a full vowel
  “He will graduate.” vs. “He is a graduate.”
- Double stress: front-stressed before noun, end-stressed when final in phrase
  “an overnight *bag” vs. “Are you staying* overnight*?”*
- -ed adjectives with matching verb past tenses
  “He is a learned *man.” vs. “He* learned *to play piano.”*
- Ambiguous abbreviations: already described in Section 14.4.1
  in, am, SAT (Saturday vs. Standard Aptitude Test)
- Borrowed words from other languages—They could sometimes be distinguishable based on capitalization.
  “El Camino Real *road in California” vs.* “real *world”*
  “polish *shoes” vs.* “Polish *accent”*
- Miscellaneous
  “The sewer *overflowed.” vs. “a* sewer *is not a tailor.”*
  “He moped *since his parents refused to buy a* moped*.”*
  “Agape *is a Greek word.” vs. “His mouth was* agape*.”*

As discussed earlier, abbreviation/acronym expansion and linguistic analysis described in Sections 14.4.1 and 14.5 are two main sources of information for TTS systems to resolve homograph ambiguities.

We close this section by introducing two special sources of pronunciation ambiguity that are not fully addressed by current TTS systems. The first one is a variation of dialects (or even personal dialect—*idiolect*). For example, some might say *tom*[*ey*]*to*, while some others might say *tom*[*aa*]*to*. Another example is that Boston natives tends to reduce the */r/* sound in sentences like “Park your ca*r* in Ha*r*va*r*d ya*r*d.” Similarly, some people use the spelling pronunciation *in-ter-es-ting* as opposed to *intristing*. Finally, speech rate and formality level can influence pronunciation. For example, the /*g*/ sound in *recognize* may be omitted in faster speech. It might be a sensible decision to output all possible pronunciations as a multiple pronunciation list and hope the synthesis back end picks the one with better acoustic/prosodic voice rendition. While true homographs may be resolved by linguistic and discourse analysis, achieving a consistent presentation of dialectal and stylistic variation is an even more difficult research challenge.

The other special source of ambiguity in TTS is somewhat different from what we have considered so far, but may be a concern in some markets. Most borrowed or foreign single words and place names are realized naturally with pronunciation normalized to the main presentation language. Going beyond that, language detection refers to the ability of a TTS system to recognize the intended language of a multiword stretch of text. For example, consider the fragment "Well, as for the next department head, that is simply *une chose entendue*." The French phrase "*une chose entendue*" (something clearly understood) might be realized in a proper French accent and phone pronunciation by a bilingual English/French reader. For a TTS system to mimic the best performance, the system must have:

- language identification capability
- dictionaries and rules for both languages
- voice rendition capability for both languages

14.7. MORPHOLOGICAL ANALYSIS

General issues in morphology are covered in Chapter 2. Here, we consider issues of relating a surface orthographic form to its pronunciation by analyzing its component morphemes, which are minimal, meaningful elements of words, such as prefixes, suffixes, and stem words themselves. This decomposition process is referred as *morphological analysis* [28]. When a dictionary does not list a given orthographic form explicitly, it is sometimes possible to analyze the new word in terms of shorter forms already present. These shorter forms may combine as prefixes, one or more stems or roots, and suffixes to generate new forms. If a word can be so analyzed, the listed pronunciations of the pieces can be combined, perhaps with some adjustment (phonological rules), to yield a phonetic form for the word as a whole.

The prefixes and suffixes are generally considered bound, in the sense that they cannot stand alone but must combine with a stem. A stem, however, can stand alone. A word such as *establishment* may be decomposed into a "stem" *establish* and a suffix *-ment*. In practice, it is not always clear where this kind of analysis should stop. That is, should a system attempt to further decompose the stem *establish* into *establ* and *-ish*? These kinds of questions ultimately belong to etymology, the study of word origins, and there is no final answer. However, for practical purposes, having three classes of entries corresponding to prefixes, stems, and suffixes, where the uses of the affixes are intuitively obvious to educated native speakers, is usually sufficient. In practical language engineering, a difference that makes no difference is no difference, and unless there is a substantial gain in compression or analytical power, it is best to be conservative and list only obvious and highly productive affixes.

The English language presents numerous genuine puzzles in morphological analysis. For example, there is the issue of abstraction: is the word *geese* one morpheme, or two (base *goose* + abstract pluralizing morpheme)? For practical TTS systems, relying on large dictionaries, it is generally best to deal with concrete, observable forms where possible. In such a lexically oriented system, the word *geese* probably should appear in the lexicon as such, with attached grammatical features including plurality. Likewise, it is simpler to include *children* in the lexical listing rather than create a special pluralizing suffix *-ren* whose use is restricted to the single base *child*.

The morphological analyzer must attempt to cover an input word in terms of the affixes and stems listed in the morphological lexicon. The covering(s) proposed must be legal sequences of forms, so that often a word grammar is supplied to express the allowable patterns of combinations. A word grammar might, for example, restrict suffixation to the final or rightmost stem of a compound, thus allowing plurality on the final element of *businessmen* but not in the initial stem (*businessesman*). In support of the word grammar, all stems and affixes in the lexicon would be listed with morphological combinatory class specifications, usually subtyped in accordance with the base POS categories of the lexicon entries. That is, verbs would typically accept a different set of affixes than nouns or adjectives. In addition, spelling changes that sometimes accompany affixation must be recognized and undone during analysis. For example, the word *stopping* has undergone final consonant doubling as part of accommodating the suffix *ing*.

A morphological analysis system might be as simple as a set of *suffix-stripping* rules for English. If a word cannot be found in the lexicon, a suffix-stripping rule can be applied to first strip out the possible suffix, including *–s*, *-'s*, *-ing*, *-ed*, *-est*, *-ment*, etc. If the stripped form can be found in the lexicon, a morphological decomposition is attained. Similarly, *prefix-stripping* rules can be applied to find prefix-stem decomposition for prefixes like *in-*, *un-*, *non-*, *pre-*, *sub-*, etc., although in general prefix stripping is less reliable.

Suffix and prefix stripping gives an analysis for many common inflected and some derived words such as *helped*, *cats*, *establishment*, *unsafe*, *predetermine*, *subword*, etc. It helps in saving system storage. However, it does not account for compounding, issues of legality of sequence (word grammar), or spelling changes. It can also make mistakes (from a synchronic point of view: *basement* is not *base* + *-ment*), some of which will have consequences in TTS rendition. A more sophisticated version could be constructed by adding elements such as POS type on each suffix/prefix for a rudimentary legality check on combinations. However, a truly robust morphological capability would require more powerful formal machinery and a more thorough analysis. Therefore, adding irregular morphological formation into a system dictionary is always a desirable solution.

Finally, sometimes in commercial product names the compounding structure is signaled by word-medial case differences, e.g., AltaVista^{TM}, which can aid phonetic conversion algorithms. These can be treated as two separate words and will often sound more natural if rendered with two separate main stresses. This type of decomposition can be expanded to find compound words that are formed by two separate nouns. Standard morphological analysis algorithms employing suffix/prefix stripping and compound word decomposition are summarized in Algorithm 14.4.

```
ALGORITHM 14.4 MORPHOLOGICAL ANALYSIS
1.  Dictionary Lookup
      Look up word w in lexicon
      If found
          Output attributes of the found lexical entry and exit
2.  Suffix Stripping
      If word ends in –s, -'s, -ing, -ed, -est, -ment, etc
          Strip the suffix from word w to form u
      If stripped form u found in lexicon
          Output attributes of the stem and suffix and exit
3.  Prefix Stripping
      If word begins with in-, un-, non-, pre-, sub-, etc
          Strip the prefix from word w to form u
      If stripped form u found in lexicon
          Output attributes of the prefix and stem and exit
4.  Compound word decomposition
      If detect word-medial case differences within word w
          Break word w into a multiple words $u_1$, $u_2$, $u_3$, …according to case changes
          For words $u_1$, $u_2$, $u_3$, goto 1.
      Else if word w can be decomposed into two nouns $u_1$, $u_2$
          Output attributes of the $u_1$, $u_2$ and exit
5.  Pass word w to letter-to-sound module
```

14.8. LETTER-TO-SOUND CONVERSION

The best resource for generating (symbolic) phonetic forms from words is an extensive word list. The accuracy and efficiency of such a solution is limited only by the time, effort, and knowledge brought to bear on the dictionary construction process. As described in Section 14.2, a general lexicon service is a critical resource for the TTS system. Thus, the first and the most reliable way for grapheme-to-phoneme conversion is via dictionary lookup.

Where direct dictionary lookup fails, rules may be used to generate phonetic forms. Under earlier naïve assumptions about the regularity and coverage of simple descriptions of English orthography, rules have traditionally been viewed as the primary source of phonetic conversion knowledge, since no dictionary covers every input form and the TTS system must always be able to speak any word. A general *letter-to-sound* (LTS) conversion is thus required to provide phonetic pronunciation for any sequence of letters.

Inspired by the phonetic languages, letter-to-sound conversion is usually carried out by a set of rules. These rules can be thought of as dictionaries of fragments with some special conventions about lookup and context. Typically, rules for phonetic conversion have mimicked phonological rewriting in phonological theory [5], including conventions of ordering, such as *most specific first*. In phonological rules, a target is given and the rewrite is indicated, with context following. For example, a set of rules that changes orthographic *k* to a velar plosive /*k*/ except when the *k* is word-initial ('[') followed by *n* might appear as:

```
k -> /sil/ % [ _ n
k -> /k/
```

The rule above reads that *k* is rewritten as (phonetic) silence when in word initial position and followed by *n*, otherwise *k* is rewritten as (phonetic) /*k*/. The underscore in the first line is a placeholder for the *k* itself in specifying the context. This little set properly treats *k* in *knight*, *darkness*, and *kitten*. These are formally powerful, context-sensitive rules. Generally a TTS system require hundreds or even thousands of such rules to cover words not appearing in the system dictionary or *exception list*. Typically rules are specified in terms of single-letter targets, such as the example for *k* above. However, some systems may have rules for longer fragments, such as the special vowel and consonant combinations in words like *neighbor* and *weigh*. In practice, a binary format for compression, a corresponding fragment matching capability, and a rule index must be defined for efficient system deployment.

Rules of this type are tedious to develop manually. As with any *expert system*, it is difficult to anticipate all possible relevant cases and sometimes hard to check for rule interference and redundancy. In any case, the rules must be verified over a test list of words with known transcriptions. Generally, if prediction of main stress location is not attempted, such rules might account for up to 70% of the words in a test corpus of general English. If prediction of main stress is attempted, the percentage of correct phonetic pronunciations is much lower, perhaps below 50%. The correct prediction of stress depends in part on morphology, which is not typically explicitly attempted in this type of simple rule system (though fragments corresponding to affixes are frequently used, such as *tion* -> /*ah ax n*/). Certainly, such rules can be made to approach dictionary accuracy, as longer and more explicit morphological fragments are included. One extreme case is to create one specific rule (containing exact contexts for the whole word) for each word in the dictionary. Obviously this is not desirable, since it is equivalent to putting the word along with its phonetic pronunciation in the dictionary.

In view of how costly it is to develop LTS rules, particularly for a new language, attempts have been made recently to automate the acquisition of LTS conversion rules. These self-organizing methods believe that, given a set of words with correct phonetic transcriptions (the offline dictionary), an automated learning system could capture significant generalizations. Among them, classification and regression trees (CART) have been demonstrated to give satisfactory performances for letter-to-sound conversion. For basic and theoretic description of CART, please refer to Chapter 4.

In the system described in [14], CART methods and phoneme trigrams were used to construct an accurate conversion procedure. All of the experiments were carried on two databases. The first is the NETALK [25], which has hand-labeled alignment between letter and phoneme transcriptions. The second is the CMU dictionary, which does not have any alignment information. The NETALK database consists of 19,940 entries, of which 14,955 were randomly selected as atraining set and the remaining 4951 were reserved for testing. Those 4951 words correspond to 4985 entries in the database because of multiple pronunciations. The hand-labeled alignments were used directly to train the CART for LTS conversion. The

CMU dictionary has more than 100,000 words, of which the top 60,000 words were selected based on unigram frequencies trained from North American Business News. Among them, 52,415 were used for training and 9719 reserved for testing. Due to multiple pronunciations, those 9719 words have 10,520 entries in the dictionary. Due to lack of alignment information, dynamic programming was used to align each letter to the corresponding phoneme before training the LTS CART.

The basic CART component includes a set of *yes-no* questions and a procedure to select the best question at each node to grow the tree from the root. The basic *yes-no* question for LTS conversion looks like "*Is the second right letter 'p'?*" or "*Is the first left output phoneme /ay/?*" The questions for letters could be on either the left or the right side. For phones, only questions on the left side were used, for simplicity. The range of question positions must be long enough to cover the long-distance phonological variations. It was found that the 11-letter window (5 for left letter context and 5 for right letter context) and 3-phoneme window for left phoneme context are generally sufficient. A primitive set of questions would be the set of all the singleton questions about each letter or phoneme identity. When growing the tree, the question that had the best entropy reduction was chosen at each node. We observed that if we allow the node to have a complex question that is a combination of primitive questions, the depth of the tree will be greatly reduced and the performance improved. For example, the complex question "*Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?*" can capture '*o*' in common suffix "*tion*" and convert it to the right phoneme. Complex questions can also alleviate the data fragment problem caused by greedy nature of the CART algorithm. This way of finding such complex questions is similar to those used in Chapter 4. The baseline system built using the above techniques has error rates as listed in Table 14.11.

Table 14.11 LTS baseline results using CART [13].

Database	Phoneme	Word
CMU Lexicon	9.7%	35.0%
NETTALK	9.5%	42.3%

The CART LTS system [14] further improved the accuracy of the system via the following extensions and refinements:

- *Phoneme trigram rescoring:* A statistical model of phoneme co-occurrence, or phonotactics, was constructed over the training set. A phonemic trigram was generated from the training samples with back-off smoothing, and this was used to rescore the *n*-best list generated by LTS.
- *Multiple tree combination:* The training data was partitioned into two parts and two trees were trained. When the performance of these two trees was tested, it was found that they had a great overlap but also behaved differently, as each had a different focus region. Combining them together greatly improved the coverage. To get a better overall accuracy, the tree trained by all the samples was used together with two other trees, each trained by half of the samples. The leaf dis-

tributions of three trees were interpolated together with equal weights and then phonemic trigram was used to rescore the *n*-best output lists.

By incrementally experimenting with addition of these extensions and refinements, the results improved, as shown in Table 14.12:

Table 14.12 LTS using multiple trees and phonemic trigram rescoring [13].

Database	Phoneme	Word
CMU Lexicon	8.2%	26.9%
NETTALK	8.1%	34.2%

These experiments did not include prediction of stress location. Stress prediction is a difficult problem, as we pointed out earlier. It requires information beyond the letter string. In principle, one can incorporate more lexical information, including POS and morphologic information, into the CART LTS framework, so it can be more powerful of learning the phonetic correspondence between the letter string and lexical properties.

14.9. EVALUATION

Ever since the early days of TTS research [21, 31], evaluation has been considered an integral part of the development of TTS systems. End users and application developers are mostly interested the end-to-end evaluation of TTS systems. This *monolithic* type of whole-system evaluation is often referred to as *black-box* evaluation. On the other hand, *modular* (component) testing is more appropriate for TTS researchers when working with isolated components of the TTS system, for diagnosis or regression testing. We often refer to this type of evaluation as *glass-box* evaluation. We discuss the modular evaluations in each modular TTS chapter, while leaving the evaluation of the whole system to Chapter 16.

For text and phonetic analysis, automated, analytic, and objective evaluation is usually feasible, because the input and output of such module is relatively well defined. The evaluation focuses mainly on symbolic and linguistic level in contrast to the acoustic level, with which prosodic generation and speech synthesis modules need to deal. Such tests usually involve establishing a test corpus of correctly tagged examples of the tested materials, which can be automatically checked against the output of a text analysis module. It is not particularly productive to discuss such testing in the abstract, since the test features must closely track each system's design and implementation. Nevertheless, a few typical areas for testing can be noted. In general, tests are simultaneously testing the linguistic model and content as well as the software implementation of a system, so whenever a discrepancy arises, both possible sources of error must be considered.

For automatic detection of document structures, the evaluation typically focuses on sentence breaking and sentence type detection. Since the definitions of these two types of document structures are straightforward, a standard evaluation database can be easily established.

In the basic level, the evaluation for the text normalization component should include large regression test databases of text micro-entities: addresses, Internet and e-mail entities, numbers in many formats (ordinal, cardinal, mathematical, phone, currency, etc.), titles, and abbreviations in a variety of contexts. These would be paired with the correct reference forms in something like the SNOR used in ASR output evaluation. In its simplest form, this would consist of a database of automatically checkable paired entries like 7% vs. *seven percent*, and $1.20 vs. *one dollar and twenty cents.* If you want to evaluate the semantic capability of text normalization, the regression database might include markups for semantic tags, so that we have 7% vs. "<number>SEVEN<ratio>PERCENT</ratio></number>", and $1.20 vs. "<money>ONE DOLLAR AND TWENTY CENTS</money>". The regression database could include domain-specific entries. This implies some dependence on the system's API—its markup capabilities or tag set. In the examples given in Table 14.13, the first one is a desirable output for domain-independent input, while the second one is suitable for normalization of the same expression in mathematical formula domain.

Table 14.13 Two examples to test domain independent/dependent text normalization.

3-4	*three to four*
	three four
<math_exp> 3-4 </math_exp>	*three minus four*

Some systems may not have a discrete level of orthographic or SNOR representation that easily lends itself to the type of evaluation described in this section. Such systems may have to evaluate their text normalization component in terms of LTS conversion.

An automated test framework for the LTS conversion analysis minimally includes a set of test words and their phonetic transcriptions for automated lookup and comparison tests. The problem is the infinite nature of language: there are always *new words* that the system does not convert correctly, and many of these will initially lack a transcription of record even to allow systematic checking. Therefore, a comprehensive test program for test of phonetic conversion accuracy needs to be paired with a data development effort. The data effort has two goals: to secure a continuous source of potential new words, such as a 24-hour newswire feed, and to maintain and construct an offline test dictionary, where reference transcriptions for new words are constantly created and maintained by human experts. This requirement illustrates the codependence of automated and manual aspects of evaluation. Different types and sources of vocabulary need to be considered separately, and they may have differing testing requirements, depending, again, on the nature of the particular system to be evaluated. For example, some systems have elaborate subsystems targeted specifically for name conversion. Such systems may depend on other kinds of preprocessing technologies, such as name identification modules, that might be tested independently.

The correct phonetic representation of a word usually depends on its sentence and even discourse contexts, as described in Section 14.6. Therefore, the adequacy of LTS conversion should not, in principle, be evaluated on the basis of isolated word pronunciations. However, a list of isolated word pronunciations is often used in LTS conversion because of its simplicity. Discourse contexts are, in general, difficult to represent unless specific applications and markup tags are available to the evaluation database. A reasonable compromise is to use a list of independent sentences with their corresponding phonetic representation for the evaluation of grapheme-to-phoneme conversion.

Error analysis should be treated as equally important as the evaluation itself. For example, if a confusability matrix shows that a given system frequently confuses central and schwa-like unstressed vowels, this may be viewed as less serious than other kinds of errors. Other subareas of LTS conversion that could be singled out for special diagnosis and testing include morphological analysis and stress placement. Of course, testing with phonemic transcriptions is the ultimate *unit test* in the sense that it contains nothing to insure that the correctly transcribed words, when spoken by the system's artificial voice and prosody, are intelligible or pleasant to hear. Phone transcription accuracy is, thus, a necessary but not a sufficient condition of quality.

14.10. Case Study: Festival

The University of Edinburgh's *Festival* [3] has been designed to take advantage of modular subcomponents for various standard functions. Festival provides a complete text and phonetic analysis with modules organized in sequence roughly equivalent to Figure 14.1. Festival outputs speech of quality comparable to many commercial synthesizers. While default routines are provided for each stage of processing, the system is architecturally designed to accept alternative routines in modular fashion, as long as the data transfer protocols are followed. This variant of the traditional TTS architecture is particularly attractive for commercial purposes (development, maintenance, testing, scalability) as well as research. Festival can be called in various ways with a variety of switches and filters, set from a variety of sanctioned programming and scripting languages. These control options are beyond the scope of this overview.

14.10.1. Lexicon

Festival employs phonemes as the basic sounding units, which are used not only as the atoms of word transcriptions in the lexicons, but also as the organizing principle for unit selection (see Chapter 16) in the synthesizer itself. Festival can support a number of distinct phone sets and it supports mapping from one to another. A phone defined in a set can have various associated phonological features, such as vowel, high, low, etc.

The Festival lexicon, which may contain several components, provides pronunciations for words. The *addenda* is an optional list of words that are unique to a particular user, document, or application. The addenda is searched linearly. The main system lexicon is expected to be large enough to require compression and is assumed to reside on a disk or other external storage. It is accessed via binary search. The lexical entry also contains POS information, which can be modified according to the preference of the system configurer. A typical lexical entry consists of the word key, a POS tag, and phonetic pronunciation (with stress and possible syllabification indicated in parentheses):

```
("walkers" N ((( w ao ) 1) (( k er z ) 0)) )
```

If the syllables structure is not shown with parentheses, a syllabification rule component can be invoked. Separate entry lines are used for words with multiple pronunciations and/or POS, which can be resolved by later processing.

14.10.2. Text Analysis

Festival has been partially integrated with research on the use of automatic identification of document and discourse structures. The discourse tagging is done by a separate component, called SOLE [11]. The tags produced by SOLE indicate features that may have relevance for pitch contour and phrasing in later stages of synthesis (see Chapter 15). These must be recognized and partially interpreted at the text analysis phrase. The SOLE tags tell Festival when the text is comparing or contrasting two objects, when it's referring to old or new information, when it's using a parenthetical or starting a new paragraph, etc., and Festival will decide, based on this information, that it needs to pause, to emphasize or deemphasize, to modify its pitch range, etc.

Additionally, as discussed in Section 14.3, when document creators have knowledge about the structure or content of documents, they can express the knowledge through an XML-based synthesis markup language. A document to be spoken is first analyzed for all such tags, which can indicate alternative pronunciations, semantic or quasi-semantic attributes (different uses of numbers by context for example), as well as document structures, such as explicit sentence or paragraph divisions. The kinds of information potentially supplied by the SABLE tags[7] is exemplified in Figure 14.7.

```
<SABLE>
<SPEAKER NAME="male1">

The boy saw the girl in the park <BREAK/> with the telescope.
The boy saw the girl <BREAK/> in the park with the telescope.

Good morning <BREAK /> My name is Stuart, which is spelled
<RATE SPEED="-40%">
<SAYAS MODE="literal">stuart</SAYAS> </RATE>
though some people pronounce it
<PRON SUB="stoo art">stuart</PRON>. My telephone number
is <SAYAS MODE="literal">2787</SAYAS>.

I used to work in <PRON SUB="Buckloo">Buccleuch</PRON> Place,
but no one can pronounce that.

</SPEAKER>
</SABLE>
```

Figure 14.7 A document fragment augmented with SABLE tags can be processed by the Festival system [3].

[7] Sable and other TTS markup systems are discussed further in Chapter 15.

For untagged input, or for input inadequately tagged for text division (<BREAK/>), sentence breaking is performed by heuristics, similar to Algorithm 14.1, which observe whitespace, punctuation, and capitalization. A linguistic unit roughly equivalent to a sentence is created by the system for the subsequent stages of processing.

Tokenization is performed by system or user-supplied routines. The basic function is to recognize potentially speakable items and to strip irrelevant whitespace or other non-speakable text features. Note that some punctuation is retained as a feature on its nearest word.

Text normalization is implemented by token-to-word rules, which return a standard orthographic form that can, in turn, be input to the phonetic analysis module. The token-to-word rules have to deal with text normalization issues similar to those presented in Section 14.4. As part of this process, token-type-specific rule sets may be applied to disambiguate tokens whose pronunciations are highly context dependent. For example, a disambiguation routine may be required to examine context for deciding whether *St.* should be realized as *Saint* or *street*. For general English-language phenomena, such as numbers and various symbols, a standard token-to-word routine is provided. One interesting feature of the Festival system is a utility for helping to automatically construct decision trees to serve text normalization rules, when system integrators can gather some labeled training data.

The linguistic analysis module for the Festival system is mainly a POS analyzer. An *n*-gram based trainable POS tagger is used to predict the likelihoods of POS tags from a limited set given an input sentence. The system uses both a priori probabilities of tags given a word and *n*-grams for sequences of tags. The basic underlying technology is similar to the work in [6] and is described in Section 14.5. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography, as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

14.10.3. Phonetic Analysis

The homograph disambiguation is mainly resolved by POS tags. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

If a word fails lexical lookup, LTS rules may be invoked. These rules may be created by hand, formatted as shown below:

(# [c h] C = /k /) // ch at word start, followed by a consonant, is /k/, e.g. Chris

Alternatively, LTS rules may be constructed by automatic statistical methods, much as described in Section 14.8 above, where CART LTS systems were introduced. Utility routines are provided to assist in using a system lexicon as a training database for CART rule construction.

In addition, Festival system employs *post-lexical rules* to handle *context coarticulation*. Context coarticulation occurs when surrounding words and sounds, as well as speech style, affect the final form of pronunciation of a particular phoneme. Examples include reduction of consonants and vowels, phrase final devoicing, and r-insertion. Some coarticulation rules are provided for these processes, and users may also write additional rules.

14.11. HISTORICAL PERSPECTIVE AND FURTHER READING

Text-to-speech has a long and rich history. You can hear samples and review almost a century's worth of work at the Smithsonian's Speech Synthesis History Project [19]. A good source for multilingual samples of various TTS engines is [20].

The most influential single published work on TTS has been *From Text to Speech: The MITalk System* [1]. This book describes the MITalk system, from which a large number of research and commercial systems were derived during the 1980s, including the widely used DECTalk system [9]. The best compact overall historical survey is Klatt's *Review of Text-to-Speech Conversion for English* [15]. For deeper coverage of more recent architectures, refer to [7]. For an overview of some of the most promising current approaches and pressing issues in all areas of TTS and synthesis, see [30]. One of the biggest upcoming issues in TTS text processing is the architectural relation of specialized TTS text processing as opposed to general-purpose natural language or document structure analysis. One of the most elaborate and interesting TTS-specific architectures is the multilingual text processing engine described in [27]. This represents a commitment to providing exactly the necessary and sufficient processing that speech synthesis requires, when a general-purpose language processor is unavailable.

However, it is expected that natural language and document analysis technology will become more widespread and important for a variety of other applications. To get an idea of what capabilities the natural language analysis engines of the future may incorporate, refer to [12] or [2]. Such generalized engines would serve a variety of clients, including TTS, speech recognition, information retrieval, machine translation, and other services which may seem exotic and isolated now but will increasingly share core functionality. This convergence of NL services can be seen in a primitive form today in Japanese *input method editors* (IME), which offload many NL analysis tasks from individual applications, such as word processors and spreadsheets, and unify these functions in a single common processor [18].

For letter-to-sound rules, NETalk [25], which describes automatic learning of LTS processes via neural network, was highly influential. Now, however, most systems have converged on decision-tree systems similar to those described in [14].

REFERENCES

[1] Allen, J., M.S. Hunnicutt, and D.H. Klatt, *From Text to Speech: the MITalk System*, 1987, Cambridge, UK, University Press.

[2] Alshawi, H., *The Core Language Engine*, 1992, Cambridge, US, MIT Press.

[3] Black, A.W., P. Taylor, and R. Caley, "The architecture of the Festival Speech Synthesis System," *3rd ESCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, University of Edinburgh pp. 147-151.

[4] Boguraev, B. and E.J. Briscoe, *Computational Lexicography for Natural Language Processing*, 1989, London, Longmans.

[5] Chomsky, N. and M. Halle, *The Sound Patterns of English*, 1968, Cambridge, MIT Press.

[6] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas pp. 136-143.

[7] Dutoit, T., *An Introduction to Text-to-Speech Synthesis*, 1997, Kluwer Academic Publishers.

[8] Francis, W. and H. Kucera, *Frequency Analysis of English Usage*, 1982, New York, N.Y., Houghton Mifflin.

[9] Hallahan, W.I., "DECtalk Software: Text-to-Speech Technology and Implementation," *Digit Technical Journal*, 1995, **7**(4), pp. 5-19.

[10] Higgins, J., *Homographs*, 2000, http://www.stir.ac.uk/celt/staff/higdox/wordlist/homogrph.htm.

[11] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia pp. 2763-2766.

[12] Jensen, K., G. Heidorn, and S. Richardson, *Natural Language Processing: the PLNLP Approach*, 1993, Boston, US, Kluwer Academic Publishers.

[13] Jiang, L., H.W. Hon, and X. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Proc. of Eurospeech*, 1997, Rhodes, Greece pp. 605-608.

[14] Jiang, L., H.W. Hon, and X.D. Huang, "(don't use) Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech'97*, 1997, Rhodes, Greece.

[15] Klatt, D., "Review of Text-to-Speech Conversion for English," *Journal of Acoustical Society of America*, 1987, **82**, pp. 737-793.

[16] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[17] Levine, J., Mason, T., Brown, D., *Lex and Yacc*, 1992, Sebastopol, CA, O'Rielly & Associates.

[18] Lunde, K., *CJKV Information Processing Chinese, Japanese, Korean & Vietnamese Computing*, 1998, O'Reilly.

[19] Maxey, H., *Smithsonian Speech Synthesis History Project*, 2000, http://www.mindspring.com/~dmaxey/ssshp/.

[20] Möhler, G., *Examples of Synthesized Speech*, 1999, http://www.ims.uni-stuttgart.de/phonetik/gregor/synthspeech/.

[21] Nye, P.W., *et al.*, "A Plan for the Fiield Evaluation of an Automated Reading System for the Blind," *IEEE Trans. on Audio and Electroacoustics*, 1973, **21**, pp. 265-268.

[22] OMF, *CML - Chemical Markup Language*, 1999, http://www.xml-cml.org/.

[23] Richardson, S.D., W.B. Dolan, and L. Vanderwende, "MindNet: Acquiring and Structuring Semantic Information from Text," *ACL'98: 36th Annual Meeting of the Assoc. for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, 1998 pp. 1098-1102.

[24] Sable, *The Draft Specification for Sable version 0.2*, 1998, http://www.cstr.ed.ac.uk/projects/sable_spec2.html.

[25] Sejnowski, T.J. and C.R. Rosenberg, *NETtalk: A Parallel Network that Learns to Read Aloud*, 1986, Johns Hopkins University.

[26] Sluijter, A.M.C. and J.M.B. Terken, "Beyond Sentence Prosody: Paragraph Intonation in Dutch," *Phonetica*, 1993, **50**, pp. 180-188.

[27] Sproat, R., *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, 1998, Dordrecht, Kluwer Academic Publishers.

[28] Sproat, R. and J. Olive, "An Approach to Text-to-Speech Synthesis" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 611-634, Elsevier Science.

[29] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, **LIX**(236), pp. 433-460.

[30] van Santen, J., *et al.*, *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag.

[31] Van-Santen, J., *et al.*, "Report on the Third ESCA TTS Workshop Evaluation Procedure," *Third ESCA Workshop on Speech Synthesis*, 1998, Sydney, Australia.

[32] Vitale, T., "An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer," *Computational Linguistics*, 1991, **17**(3), pp. 257-276.

[33] W3C, *Aural Cascading Style Sheets (ACSS)*, 1997, http://www.w3.org/TR/WD-acss-970328.

[34] W3C, *W3C's Math Home Page*, 1998, http://www.w3.org/Math/.

[35] W3C, *Extensible Markup Language (XML)*, 1999, http://www.w3.org/XML/.

[36] Wall, L., Christiansen, T., Schwartz, R., *Programming Perl*, 1996, Sebastopol, CA, O'Rielly & Associates.